第3章

面向对象技术

【教学提示】

在日常生活中，很多事情需要事先设计好模板并进一步制作，比如制作衣服首先需要一份设计图，根据图样来裁剪布料制作。在Java语言中，类好比日常生活中描述的一个物品，而对象好比制作完成后实实在在的东西。本章将介绍面向对象的基本概念，类的定义，类成员变量的定义和方法的定义，方法参数等知识。

本章主要内容：

- 面向对象的基本概念；
- 类的定义；
- 类的主方法和构造方法；
- 成员变量和局部变量的区别；
- 对象的生命周期；
- 匿名对象。

3.1 面向对象的基本概念

面向对象思想最初起源于20世纪60年代中期的仿真设计语言Simula Ⅰ。它模拟现实世界中的事物，把软件系统抽象成各种对象的集合，以对象为最小系统单位，这种思想更接近人类的自然思维，给程序开发人员更加灵活的创造思维空间。

3.1.1 面向对象程序设计思想

传统的程序采用结构化的程序设计思想，即面向过程的程序设计思想，典型的编程语言即C语言，针对某一需求，自顶向下，逐步细化，将需求通过模块的形式来实现，然后对模块中的问题进行结构化编码。这种方式是针对具体问题来求解，然而当用户的需求不断增加，软件规模不断变大，传统的面向过程的开发方式逐渐暴露出许多不足，如软件开发周期长、后期工程维护困难等。20世纪80年代后期，人们提出了面向对象(Object Oriented Programming，OOP)的程序设计方式。面向对象程序设计方法是指用面向对象的方法指导整个程序设计的全过程。所谓面向对象是指以对象为中心，分析、设计及构造应用程序的机

制，将数据和数据处理的方法紧密地结合在一起形成类，而再将类实例化，就形成对象。在面向对象的程序设计思想中，不再需要考虑数据结构和功能函数，只要关注对象就可以了。

举一个简单的例子便于读者理解这两种编程方式。假如想做烤鸡，按照传统的做法，是将一只鸡处理干净后，在里、外面都刷上适当的调味料，静置几个小时后，用烤架架起放在烤箱中，设置好温度进行烤制，并在烤制过程中每间隔一段时间翻转并刷调料，直至烤熟为止，这种做法就是面向过程模式。但如果制造出一台新型的全自动化“烤鸡”机器，只要在这个机器指定的入口扔一只鸡进去，这台机器运作后就会在另一个出口输送出一只香喷喷的烤鸡，这种做法就是面向对象模式。很明显，面向过程的烤鸡方法，采用的是按顺序按步骤制作。我们将这些步骤抽取出关键词对应编程的术语，每个关键词代表一个功能函数：拔毛→清洗→刷调味料→静置→架烤架→翻转→刷调味→取出。再来看面向对象方式制作的烤鸡：制造机器→放入鸡→取出烤鸡，就这么简单，不必理会机器内部是怎么运作的，只要知道这台机器的用途是制作烤鸡，操作方法是往入口丢一只鸡进去，再从出口拿这只鸡出来就行。面向过程方式不用设计机器，操作起来比较便利，但如果要进行量产，面向对象方法的优势就显现出来了。这里的自动化机器的制作，就好比定义面向对象程序设计中的类(Class)，制作成的每只鸡可看作一个对象，在机器内部来设定上面所述的那些功能方法(行为)。而在机器中预先加入固定配比的调料(属性)，可以得到各种口味的烤鸡(对象)。由此可见，对象还具有属性和行为。在面向对象程序设计中，使用属性来描述对象的状态，使用方法来处理对象的行为。

3.1.2 面向对象程序设计的特点

面向对象编程更加符合人的思维模式，可以高质量、高效率地编写程序，并且程序后期易维护、易扩展。更重要的是，面向对象程序设计更有利于系统开发时的责任分工，能有效地组织和管理一些比较复杂的大型应用程序开发。面向对象程序设计的特点主要有封装性、继承性和多态性。

1. 封装性

封装是面向对象的主要特征之一，是一种信息隐蔽技术，它体现于类的说明，是对象的重要特性。封装使数据和加工该数据的方法(函数)封装为一个整体，以实现独立性很强的模块，使得用户只能见到对象的外特性(对象能接收哪些消息，具有哪些处理能力)，而对象的内部特性(保存内部状态的私有数据和实现加工能力的算法)对用户是隐蔽的。封装的目的在于把对象的设计者和对象的使用者分开，使用者不必知晓行为实现的细节，只需用设计者提供的消息来访问该对象。

2. 继承性

继承性是子类自动共享父类的非私有的数据和方法的机制。它由类的派生功能体现。一个类直接继承其他类的全部描述，同时可修改和扩充。继承具有传递性。继承分为单继承(一个子类只有一个父类)和多重继承(一个子类有多个父类)。类的对象是各自封闭的，如果没有继承性机制，则类对象中数据、方法就会出现大量重复，继承不仅支持系统的可重用性，而且还促进系统的可扩充性。例如，已经存在手机类，该类中包括两个方法，分别是接听电话 call()和发短信 message()，现在再定义一个智能手机类，就可以先继承手机类，实现

手机类的两个方法，然后再创建两个新的方法，包括拍照方法 photograph()和玩游戏的方法 playgame()，如图 3-1 所示。由此可见，继承简化了新类的设计。

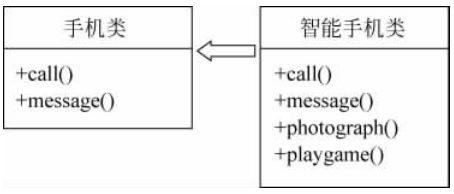

图 3-1　手机类和智能手机类的类图

3. 多态性

对象根据所接收的消息而做出动作。同一消息为不同的对象接收时可产生完全不同的行动，这种现象称为多态性。利用多态性用户可发送一个通用的信息，而将所有的实现细节都留给接收消息的对象自行决定，因此，同一消息即可调用不同的方法。例如，Print 消息被发送给一个图或表时调用的打印方法与将同样的 Print 消息发送给一个正文文件而调用的打印方法会完全不同。多态性的实现受到继承性的支持，利用类继承的层次关系，把具有通用功能的协议存放在类层次中尽可能高的地方，而将实现这一功能的不同方法置于较低层次，这样，在这些低层次上生成的对象就能给通用消息以不同的响应。例如，定义一个动物类，类中存在一个动物行为叫声 shout()，接着定义两个动物类的子类——猫和狗，这两个类都重写了动物类中的 shout()方法，实现了自己的叫声行为的处理，如图 3-2 所示。

由此可见，当动物类执行 shout()时不用判断执行哪个类的 shout()方法，因为 Java 编译器会自动根据所传递的参数进行判断，根据运行时对象类型的不同执行不同的操作。多态性丰富了对象的内容，扩大了对象的适应性。

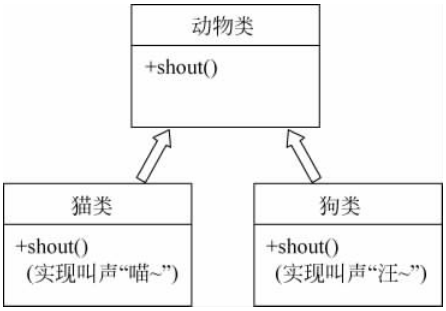

图 3-2　动物类之间的继承关系

3.2　类

Java 语言是面向对象的程序设计语言，引入类和对象的概念，类是用来创建对象的模板，它包含被创建对象的属性和方法的定义。因此要学习 Java 编程就必须要学会类的定义方法，即怎样用 Java 的语法去描述一类事物共有的属性和行为。而对象的属性通过变量来

刻画，也就是类的成员变量，而对象的行为通过方法来实现，也就是成员方法。方法可以操作属性形成一定的算法来实现一个具体的功能。类把属性和方法封装成一个整体。

3.2.1 类的定义

类是 Java 的核心，所有的 Java 程序都是基于类的。类是一个抽象的东西，描述的是一个物品的完整信息，比如房子和图纸的关系，在 Java 里，图纸就是类，定义了房子的各种信息，而房子是类的实体。

1. 类的一般结构

定义类也可称为声明类，它的主要内容包括成员变量和成员方法，成员变量描述的是对象的属性，成员方法描述的是对象的行为。下面先介绍类的一般结构，类是通过关键字 class 来定义的，在 class 后面加上类的名称，就创建了一个类，在类里面可以定义类的成员变量和成员方法，类的一般结构如下。

```
[类修饰符] class 类名称
{
//～～～～～～～～～声明成员变量～～～～～～～～～～～～～
  [修饰符] 数据类型  成员变量名称;
  …
  //～～～～～～～～声明成员方法～～～～～～～～～～～～～
  [修饰符] 返回值的数据类型 方法名(参数 1,参数 2,…)
  {
    语句序列;
    return [表达式]; }
    …
}
```

其中，class 前方的类修饰符是可选项，用来说明类的特殊性质，具体修饰符的含义如表 3-1 所示。

表 3-1 类修饰符

修饰符	含　义
public	将一个类声明为公共类，它可以被任何对象访问，一个程序的主类必须是公共类(唯一的访问控制符)
abstract	将一个类声明为抽象类，没有实现的方法，需要子类提供方法的实现
final	将一个类声明为最终类即非继承类，表示它不能被其他类所继承
friendly	友元类型，默认的修饰符，只有在相同包中的对象才能使用这样的类

2. 类的成员变量

Java 语言用成员变量来表示类的状态和属性，声明成员变量的基本语法格式如下：

```
[修饰符] 变量类型 变量名 [ = 初值]
```

其中，[]中的修饰符是可选项，修饰符的含义如表 3-2 所示。

表 3-2 成员变量的修饰符含义

修饰符	含　义
public	公共访问控制符。指定该变量为公共的,它可以被任何对象的方法访问
private	私有访问控制符。指定该变量只允许自己类的方法访问,其他任何类(包括子类)中的方法均不能访问此变量
protected	保护访问控制符。指定该变量可以被它自己的类及其子类访问,在子类中可以覆盖此变量
friendly	默认的友元访问控制符,在同一个包中的其他类可以访问此变量,而其他包中的类不能访问该变量
final	最终修饰符。指定此变量的值不能改变
static	静态修饰符。指定该变量被所有对象共享,即所有的实例都可使用该变量

其中,成员变量的访问控制符 private 和 protected 可以组合在一起使用,一个成员变量可以被两个以上的修饰符同时修饰,但有些修饰符是不能同时定义在一起的。在定义类的成员变量时,可以同时赋初值,表明成员变量的初始状态,但对成员变量的操作只能放在方法里。

3. 类的成员方法

Java 中类的行为由类的成员方法来实现。声明方法的语法格式如下:

```
[修饰符] 返回值的数据类型 方法名(参数 1,参数 2,…)
{
    语句序列;
    return [表达式];
}
```

方法体

其中,[]中的修饰符是可选项,成员方法修饰符含义如表 3-3 所示。

表 3-3 成员方法的修饰符含义

修饰符	含　义
public	公共访问控制符。指定该方法为公共的,它可以被任何对象的方法访问
private	私有访问控制符。指定该方法只允许自己类的方法访问,其他任何类(包括子类)中的方法均不能访问此方法
protected	保护访问控制符。指定该方法可以被它的类及其子类访问
friendly	默认的友元访问控制符,在同一个包中的其他类可以访问此方法,而其他包中的类不能访问该方法
final	最终修饰符。指定该方法不能被重载
static	静态修饰符。指定不需要实例化一个对象就可以激活的方法
synchronized	同步修饰符。在多线程程序中,该修饰符用于在运行前对它所属的方法加锁,以防止其他线程访问,运行结束后解锁
native	本地修饰符。指定此方法的方法体是用其他语言(如 C)在程序外部编写的

成员方法允许 private 和 protected 组合在一起来修饰一个方法;其他修饰符都只有一个。

【例 3-1】 Student 类创建。

程序分析:创建一个类 Student,定义成员变量和成员方法。

源代码：

```
//Student.java[例 3-1]
//开发者：唐璐
//基础 Student 类创建
public class Student {
String Sid;                    //学号
String Sname;                  //姓名
String Ssex;                   //性别
int Sage;                      //年龄
public void Introduce(){       //定义 Introduce()方法
    System.out.println("学号为" + Sid + "的学生的名字叫" + Sname + ",今年" + Sage + "岁.");
}
}
```

运行该程序没有显示错误代码，因此该定义符合要求，但因为没有主方法，程序没有运行入口，不会产生运行结果。

3.2.2 类的主方法

主方法是类的入口，它定义了程序从何处开始，主方法提供对程序流向的控制，Java 编译器通过主方法来执行程序，主方法的语法如下：

```
public static void main(String[] args){
      //方法体
}
```

在主方法的定义中可以看出主方法具有以下特点。

(1) 方法的修饰符是 public，说明 main()方法可以被外部调用。

(2) 主方法必须由 static 修饰，因此主方法是静态的，如果直接在主方法中调用其他方法，则该方法必须也是静态的。

(3) 主方法没有返回值。

(4) 主方法的形式参数为数组。其中，args[0]～args[n]分别代表程序的第 1～n 个参数，可以使用 args.length 获取参数的个数。

应用程序一般只有一个主方法，下面编写一个小程序向类传递参数。

【例 3-2】 主方法向类传递参数。

程序分析：主方法 main()接收一个 String 类型的数组参数，该数组保存执行 Java 命令时传入的参数。

源代码：

```
//MainDemo.java[例 3-2]
//开发者：唐璐
//主方法向类传入参数
public class MainDemo {
public static void main(String[] args) {
  System.out.println("由主方法向类传入参数演示程序：");
   for (int i = 0;i < args.length ;i++){     //循环打印传入的参数
```

```
            System.out.println("传入第" + (i + 1) + "个参数: " + args[i]);
        }
    }
}
```

为了能清晰地看到参数传递的结果，在 MyEclipse 中设置三个参数，在项目包根目录上单击右键选择 Run as→Run Configurations，在弹出的对话框中选择第二个选项卡 Arguments，填写参数，多个参数用空格隔开，单击 Run 按钮运行，如图 3-3 所示。

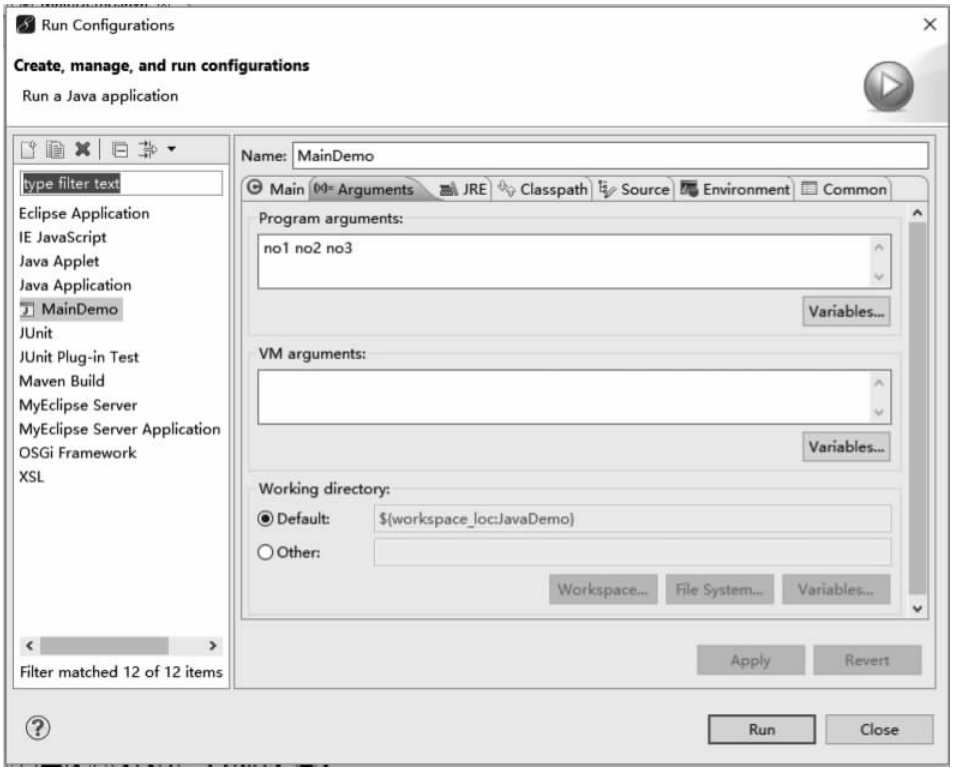

图 3-3　MyEclipse 中设置参数

执行并运行程序，如图 3-4 所示。

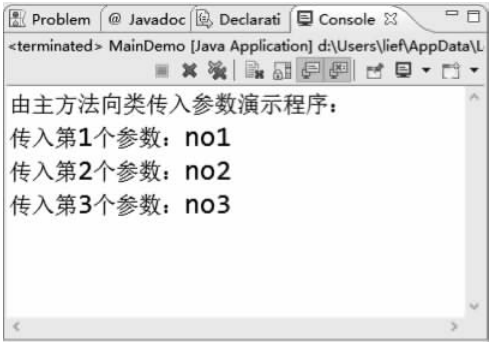

图 3-4　例 3-2 的运行结果

3.2.3 成员变量与局部变量

由类和方法的定义可知，类和方法都可以定义字的变量，在类中定义的变量称作成员变量，而在方法中定义的变量称为局部变量，成员变量和局部变量是有一定区别的，具体主要有以下 4 个方面区别。

1. 从语法形式上看

(1) 成员变量是属于类的，而局部变量是方法中定义的变量或方法的参数；

(2) 成员变量可以被 public、private、static 等修饰符所修饰，而局部变量则不能被访问控制修饰符及 static 所修饰；

(3) 成员变量和局部变量都可以被 final 所修饰。

2. 从变量在内存中的存储方式上看

成员变量是对象的一部分，而对象是存在于堆内存的，局部变量是存在于栈内存的。

3. 从变量在内存中的存在时间上看

成员变量随着对象的创建而存在，而局部变量随着方法的调用而产生，随着方法的调用结束而自动消失。

4. 从自动赋值上看

成员变量如没被赋值，则会自动以类型的默认值赋值，而局部变量在参与编译运算中必须赋初值。

【例 3-3】 成员变量和局部变量赋初始值实例。

程序分析：在程序 3-1 中，定义的成员变量均没有赋初始值，但编译没有出错，如果在 main 方法中添加两个局部变量，查看编译运行结果。

源代码：

```
//MainDemo01.java[例 3 - 3]
//开发者：唐璐
//测试局部变量是否需要赋初值
public class MainDemo01 {
int m;
public static void main(String[ ] args) {
         int n;
         String j = "I Love Java!";
               System.out.println(n);
               System.out.println(j);
    }
}
```

执行并运行，由于整型数据 n 是 main()方法的局部变量，局部变量不能由系统自动赋初值，因此编译出错，产生错误代码如下。

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    The local variable n may not have been initialized
      at MainDemo01.main(MainDemo01.java:7)
```

3.2.4 类的构造方法

在类中除了成员方法之外，还存在一种特殊类型的方法，那就是构造方法。构造方法的作用是对对象中的所有成员变量进行初始化，在创建对象时立即被调用。构造方法是一种特殊的方法，它的名字必须与它所在类的名字完全相同，构造方法的语法格式如下：

```
类的修饰符 类的名称(参数列表)
{
   //方法体
}
```

(1) 构造方法可以使用的修饰符有 public、protected、default、private，默认为 default 类型。

(2) 构造方法的名称必须要和类的名称相同。

(3) 构造方法没有返回值，void 也不能写。

(4) 构造方法的参数可有可无，可以有一个参数，也可以有多个参数。

(5) 构造方法一般不能由编程人员显式使用，而是用 new 来调用。

【例 3-4】 错误的构造方法建立。

程序分析：下面的代码演示了一个错误的构造方法的建立。

源代码：

```
//GouZaoDemo01.java[例 3 - 4]
//开发者：唐璐
//错误的构造方法建立
public class GouZaoDemo01 {
//创建一个构造方法
public void GouZaoDemo01()
    {
    //方法体
    }
    }
```

代码解析：这段代码在编译时和运行时都没有错误，因为 GouZaoDemo01()在这里不被认为是构造方法，而是一种普通方法，本例中在去掉 void 后，GouZaoDemo01()则成为构造方法。

构造方法可以有多种形式，其中主要分为有参构造方法和无参构造方法，在有参构造方法中还可以分为一个参数和多个参数的形式，其中参数类型可以相同，也可以不同。在一个类中可以有多个构造方法，下面的程序就演示了多种不同的构造方法。

```
public class House {
     House(){
        //构造方法方法体
        }
     House(String a){
        //构造方法方法体
        }
     House(String b,String c){
```

```
        //构造方法方法体
        }
    House(String d,int e){
        //构造方法方法体
        }
}
```

如果一个类在定义时没有显式定义构造方法，并不意味着这个类没有构造方法，而是默认生成一个没有方法体的空的构造方法，具体格式如下：

```
类名()
  {
  }
```

如果类被定义为 public，则默认构造方法前面也会加 public 修饰符，一旦用户为该类显式地定义构造方法后，系统就不会再提供默认的构造方法，这是由于 Java 的覆盖(Overriding)所致。关于覆盖的概念会在以后的章节中介绍。默认构造方法用来实现成员变量的初始化，Java 语言中各种类型变量的初始值如表 3-4 所示。

表 3-4 Java 语言中各种类型变量的初始值

类型	初始值
byte	0
short	0
int	0
float	0.0F
long	0L
double	0.0D
char	'\u0000'
boolean	false
引用类型	null

3.3 对象

对象是用于模拟现实世界中的实体程序元素，如果将需要解决的问题称为问题空间，将解决问题的程序称为解空间，在问题空间中的实体就会被映射成解空间中的对象。从面向对象程序设计的思想来看，对象是程序操作的基本单位，所谓的程序运行就是对象之间不断相互驱动、相互作用的过程。因此，在定义类之后，需要将类进行实例化，即创建对象。在 Java 语言中对象属于引用类型的变量，一个对象的生命周期需要经历创建、使用、销毁三个阶段。

3.3.1 对象的创建

细心的读者一定会发现，例 3-1 编译正确但是没有运行结果，的确，单纯构建一个类是没有任何功能的，它仅仅如同建立了一个图纸，不进行下一步实例化，只能纸上谈兵。那应

该怎么办呢？正确做法是利用这些类创建若干对象实例，让这些对象实例相互协作完成指定功能。

创建对象的步骤如下。

(1) 声明指向“由类所创建的对象”的变量；

(2) 利用 new 运算符创建新的对象，并指派给前面所创建的变量。

由上面的步骤可以得到创建对象的第一种格式：

```
类名 对象名称;                    //声明对象
对象名称 = new 类名( );           //实例化对象
                                  //创建对象使用 new 创建,而类名()是该类的某一个构造方法
```

例如，要创建学生类 Student 的对象 stu，可用下面的语句来创建：

```
Student stu;                      //声明指向对象的变量 stu
stu = new Student();              //利用 new 创建新对象,并让变量 stu 指向它
```

通过这两个步骤，就可以利用指向对象的变量 stu，访问到“类 Student 创建的对象”的内容，即成员变量或成员方法。另外，在创建对象时也可以将上面两个语句合并成一行书写，即在声明对象的同时使用相应的构造方法来进行初始化对象。

创建对象的第二种格式：

```
类名 对象名称 = new 类名(); //在声明对象的同时实例化对象
```

因此要创建学生类 Student 的对象 stu，也可用下面的语句来创建：

```
Student stu = new Student();
```

对于新创建的对象 stu，由于是从类 Student 产生的，所以具有可以用来保存设置该类中成员变量和成员方法的功能。

下面介绍对象在创建过程中，内存是怎样分配的。类是一种引用数据类型，这与数组是一样的。Student 类有 4 个属性，Sid 属性、Sname 属性、Ssex 属性和 Sage 属性，而属性是需要内存来存储的。下面以 stu 对象为例来讲解。执行 Student stu;时，Java 系统会在栈内存开辟一个空间给对象 stu，如图 3-5 所示。

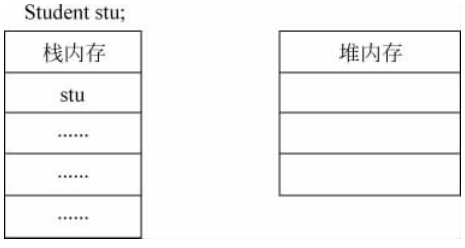

图 3-5 声明对象内存分配

程序继续向下执行到 stu＝new Student();时，Java 系统会在栈内存开辟空间保存对象 stu 属性的内容，如图 3-6 所示。

stu 变量本身存储的只是一个地址值，没有存储任何实际数据，但它指向了 Student 对象。所以访问 stu 对象的属性和方法时，实际上是访问 stu 所引用对象的属性和方法。

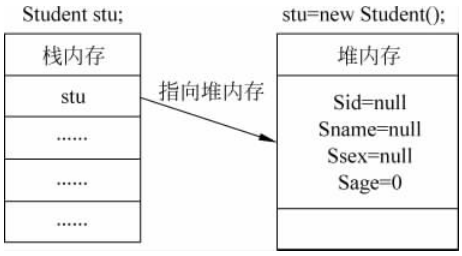

图 3-6 实例化对象内存分配

3.3.2 对象的使用

创建新的对象之后，就可以对对象的成员或成员方法进行访问。通过运算符“.”实现访问，通过对象来引用对象成员或成员方法的格式如下：

引用对象成员格式：<对象名>.<对象成员>

引用成员方法格式：<对象名>.<成员方法>

如果引用的是成员方法，只要在成员方法名的圆括号内提供所需参数即可，如果方法不需要参数，则用空括号。

例如：

```
stu.Sname = "tina";
stu.Sage = 19;
stu.Introduce();
```

【例 3-5】 建立对象并调用成员变量和成员方法。

程序分析：定义一个类 Student，再定义一个类 Demo01，在 Demo01 中创建类 Student 的两个对象 stu1 和 stu2，利用这两个对象调用相应的 Student 类的对象成员和成员方法，实现打印学生的简单介绍。

源代码：

```
//Demo01.java[例 3 - 5]
//开发者：唐璐
//建立对象并调用成员变量和成员方法
class Student {
String Sname;                          //姓名
String Ssex = "男";                    //性别赋初始值"男"
int Sage;                              //年龄
public void Introduce(){               //定义 Introduce()方法
    System.out.println("小" + Ssex + "孩" + Sname + ",今年" + Sage + "岁.");
}
}
public class Demo01 {
    public static void main(String[] args) {
    Student stu1 = new Student();      //创建对象 stu1
    Student stu2 = new Student();      //创建对象 stu2
    1.  stu1.Sname = "李利";           //利用 stu1 设置"李利"的信息
    2.  stu1.Sage = 19;
```

```
    3.  stu1.Introduce();              //利用 stu1 调用成员方法
    4.  stu1.Sname = "刘星";           //利用 stu1 设置"刘星"的信息
    5.  stu1.Sage = 20;
    6.  stu1.Introduce();              //利用 stu1 调用成员方法
    7.  stu2.Sname = "孙明明";         //利用 stu2 设置"孙明明"的信息
    8.  stu2.Sage = 19;
    9.  stu2.Introduce();              //利用 stu2 调用成员方法
    10. stu2.Sname = "唐琳琳";         //利用 stu2 设置"唐琳琳"的信息
    11. stu2.Ssex = "女";              //再次设置 Ssex 的值
    12. stu2.Sage = 18;
    13. stu2.Introduce();              //利用 stu1 调用成员方法
    14. stu1.Sname = "李楠";           //利用 stu1 设置"李楠"的信息
    15. stu1.Sage = 19;
    16. stu1.Introduce();              //利用 stu1 调用成员方法
    }
}
```

代码解析：Student 类中定义了三个成员变量：整型 Sage、字符串类型 Sname 和 Ssex，Ssex 赋初值“男”，同时在该类定义了成员方法 Introduce()，该方法是无返回值类型，内容是直接打印学生的介绍。语句段 1～3 和语句段 4～6 可以看出一个对象可以反复设置多次对象调用。

虽然在类定义中设置了 Ssex 变量的初始值，但其值还是可以被修改的，如语句 11 将 Ssex 值设置成“女”，则通过 stu1 调用的成员方法输出为“小女孩”，但语句 16 输出则仍然为“小男孩”，这说明这两个对象的成员是分配在不同的内存块中，因此修改一个对象的 Ssex 值，不影响另外一个对象的 Ssex 值。

执行并运行，当 Demo01.java 编译成字节码文件之后，会生成 Demo01.class 和 Student.class 两个字节码文件，并且 Java 会把它们放在同一个目录下，这是因为 Demo01.java 源文件中定义了两个类(Demo01 和 Student)的原因，由此可知，Java 源文件中定义多个类，经过编译之后就会产生数目相等的.class 文件。运行结果如图 3-7 所示。

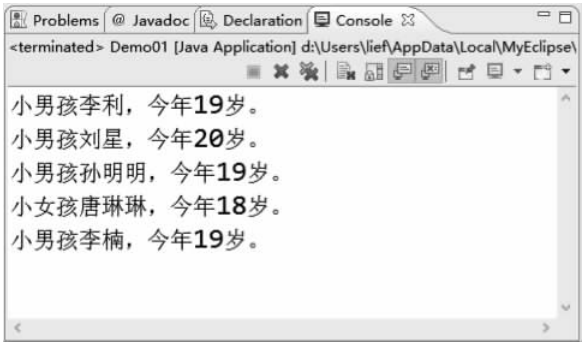

图 3-7　例 3-5 的运行结果

3.3.3　对象的销毁

在许多程序设计语言中，需要手动释放对象所占用的内存，但在 Java 中则不需要手动完成这项工作。它提供了几个方法来进行对象的销毁。

垃圾回收机制是Java平台中使用最频繁的一种对象销毁方法。垃圾回收器会全程监测Java应用程序的运行情况。当有些对象成为垃圾时，垃圾回收器就会销毁这些对象，并释放这些对象所占用的内存空间。在这里，程序开发职员只需要知道，在哪些情况下垃圾回收器会以为这些对象是垃圾对象。通常情况下，如果发生以下两种情况时，系统会以为这些对象是垃圾对象，需要销毁。

(1) 将一个NULL值赋值给对象。如用户先建立了一个对象str1，对象用完了之后，再利用赋值语句，将NULL值赋值给这个对象str1，即使用str1＝NULL的方式将NULL值赋值给这个对象。此时这个对象与内存中对象的存储地址之间就失去了联系。这个对象就会被视为垃圾对象，就会被垃圾回收器销毁。

(2) 对象超出了作用范围，这个对象就被以为是垃圾对象，被垃圾回收器回收并释放内存。

另外，Java提供了一个名为finalize()的方法，通过这个方法可以显式地让系统回收垃圾对象。它的工作原理是一旦垃圾收集器准备好释放对象占用的存储空间，它首先调用finalize()，而且只有在下一次垃圾收集过程中，才会真正回收对象的内存。这是一个Object类的方法。通常情况下，这个方法被声明为protected。程序开发职员在必要的时候，可以在自定义的类中定义这个方法，注意每个类有且只有一个finalize()方法。

3.4 类中的方法

前面简单介绍了类和对象的创建和使用方法，所涉及的方法都是无返回值或无参数传递的方法。下面具体介绍一下在类中成员方法之间的调用，方法的返回值及带参数的成员方法的调用。

3.4.1 类自身成员方法之间的相互调用

在前面例子中所简述的方法都是在本类的外部被调用，但实际上，在类定义的内部，方法与方法之间也可以相互调用。

【例3-6】 在本类中方法之间相互调用。

程序分析：定义一个课程类Course，该类中有两个成员变量Cname和credit，两个成员方法CHour()和Export()，并建立测试类Demo02，在测试类中建立Course的对象，进行调用运行。

源代码：

```
//Demo02.java[例 3－6]
//开发者：唐璐
//在本类中方法之间相互调用
class Course{                          //建立课程类
        String Cname;                  //课程名
        int credit;                    //学分
        int CHour()                    //创建计算学时的方法
        {
```

```
        return credit * 16;          //在类声明内可以直接使用成员 credit 的名称
    }
    void Export(){                   //创建无返回值的输出方法
    System.out.println(Cname + CHour() + "学时");     //在类内调用 CHour()方法
  }
}

public class Demo02 {
    public static void main(String[] args){
        Course course = new Course();
        course.Cname = "Java 程序设计";
        course.credit = 3;
        course.Export();             //调用成员方法
    }
}
```

代码解析：在 CHour()中直接使用了 Course 类的 credit 变量，说明在类声明的内部可以直接使用本类的成员变量。而在类声明外（如 main()方法）用到成员变量时，必须指明是哪个类的成员变量，也就是要用“指向对象的变量.成员名”的语法来访问对象中的成员。

在 Export()方法中，直接使用了 CHour()方法，因此可以看出，在同一个类的定义里面，某一个方法可以直接调用本类的其他方法，而不需要加上“对象名”。

执行并运行，运行结果如图 3-8 所示。

图 3-8　例 3-6 的运行结果

3.4.2　类中定义带参数的成员方法

调用方法并传递参数时，参数其实就是方法的自变量，所以参数要放在方法的括号内来进行传递。括号内的参数可以是数值型，字符串型，甚至是对象。

【例 3-7】 定义带有参数的成员方法。

程序分析：修改例 3-6，将 Course 类中的成员方法修改为能够接收两个参数的方法，并实现输出打印课程名和学时。

源代码：

```
//Demo03.java[例 3 - 7]
//开发者：唐璐
//定义带有参数的成员方法
class Course{                        //建立课程类
        String Cname;                //课程名
        int credit;                  //学分
        void CHour(String c,int i)   //创建带参数的方法，实现给两个变量赋值
```

```
        {     Cname = c;
              credit = i;
              System.out.println(Cname + "课程的学时是" + credit * 16);
        }
    }

public class Demo03{
    public static void main(String[] args){
        Course course = new Course();
        course.CHour("Java 程序设计",3);
    }
}
```

代码解析：当执行 course.CHour("Java 程序设计",3);语句时，Course 类中的 CHour 方法就会接收传递过来的两个参数，并将这两个参数赋值给类中相应的成员变量。

运行结果详见例 3-6。

3.4.3 类中定义带参数的构造方法

构造方法的作用是对类进行初始化，可以定义能够接收参数的构造方法，在对象建立时对成员变量进行赋值。

【例 3-8】 定义带有参数的构造方法。

程序分析：修改例 3-6，在 Course 类中建立能够接收两个参数的构造方法，并实现输出打印课程名和学时。

源代码：

```
//Demo04.java[例 3 - 8]
//开发者：唐璐
//定义带有参数的构造方法
class Course{                          //建立课程类
        String Cname;                  //课程名
        int credit;                    //学分
        Course(String c,int i)         //创建带两个参数的构造方法
        {    Cname = c;
             credit = i;
             System.out.println(Cname + "课程的学时是" + credit * 16);
        }
    }
public class Demo04 {
    public static void main(String[] args){
        Course course = new Course("Java 程序设计",3);     //利用带参构造方法创建对象
    }
}
```

代码解析：当创建对象时，构造方法会接收两个参数，并将这两个参数赋值给类中相应的成员变量，然后继续执行构造方法中的输出语句。

运行结果详见例 3-6。

3.5 匿名对象

当一个对象被创建之后，在调用该对象的方法时，也可以不定义对象的引用变量，而直接调用这个对象的方法，这样的对象叫作匿名对象。

【例 3-9】 匿名对象。

程序分析：定义一个类 Student，再定义一个类 AnonymousDemo，在 AnonymousDemo 中创建类 Student 的两个对象，即 stu1 和 stu2，利用这两个对象调用相应的 Student 类的对象成员和成员方法，实现打印学生的简单介绍。

源代码：

```
//AnonymousDemo.java[例 3-9]
//开发者：唐璐
//建立对象并调用成员变量和成员方法
class Course{                          //建立课程类
    String Cname;                      //课程名
    int credit;                        //学分
    int CHour(int cr)                  //创建计算学时的方法
    {     return cr * 16;
    }
}
public class AnonymousDemo{            //匿名类测试
    public static void main(String[] args){
        System.out.print("学分是：");
        //利用匿名对象的用法，输出学时数，不用创建对象
        System.out.print(new Course().CHour(3));
    }
}
```

代码解析：这里的 Course()就是匿名对象，这个语句没有产生任何对象，而是直接用 new 运算符创建了 Course 类的对象并直接调用了它的 CHour(int cr)方法。这个语句执行完后，这个对象就成为垃圾。

执行后得到运行结果：

```
学分是：48
```

使用匿名对象通常有如下两种情况。

(1) 如果对一个对象只需要进行一次方法调用，那么就可以使用匿名对象。

(2) 将匿名对象作为实参传递给一个方法调用。

```
public static void getSomeOne(MyClass c)
{ … }
```

可以用下面的语句调用：

```
getSomeOne( new MyClass());
```

小结

本章主要讲解了面向对象程序设计的基本概念，介绍了类和对象的创建和使用。通过本章的学习，读者应该对 Java 语言的类和对象的编程思想有了一定认识，可以编写较简单的语句结构。

习题

1. 填空题

(1) 类中可以定义(　　)和(　　)。

(2) 在 Java 中创建一个对象的关键字是(　　)。

(3) 在 Java 中定义类的关键字是(　　)。

2. 选择题

(1) 有一个类 Demo，对其构造方法的正确声明是(　　)。

A. void Demo(int x){…}　　B. Demo(int x){…}

C. Demo Demo(int x){…}　　D. int Demo(){} Java

(2) 下列构造方法的调用方式中，正确的是(　　)。

A. 如一般方法调用　　B. 由 new 运算符自动调用

C. 由用户直接调用　　D. 由系统调用

(3) 不允许作为类及类成员的访问控制符的是(　　)。

A. public　　B. private

C. static　　D. protected

3. 简答题

(1) 类与对象的区别是什么？

(2) 成员变量与局部变量有什么区别？

(3) 什么叫匿名对象？一般在什么情况下使用匿名对象？

4. 编程题

请按要求定义一个 Student 类，并进行测试。

要求如下：

(1) Student 类中包含姓名，性别，年龄和成绩 4 个属性。

(2) 分别给这两个属性定义两个方法，一个方法用于设置年龄值，另一个方法用来获取成绩值。

(3) Student 类中定义一个无参的构造方法和一个接收两个参数的构造方法，两个参数分别是姓名和性别属性。

(4) 在测试类中创建两个 Student 类的对象，一个对象调用无参构造方法，并且用这个对象调用(2)中的两个方法给年龄和成绩两个成员变量赋值；另一个对象调用含两个参数的构造方法。