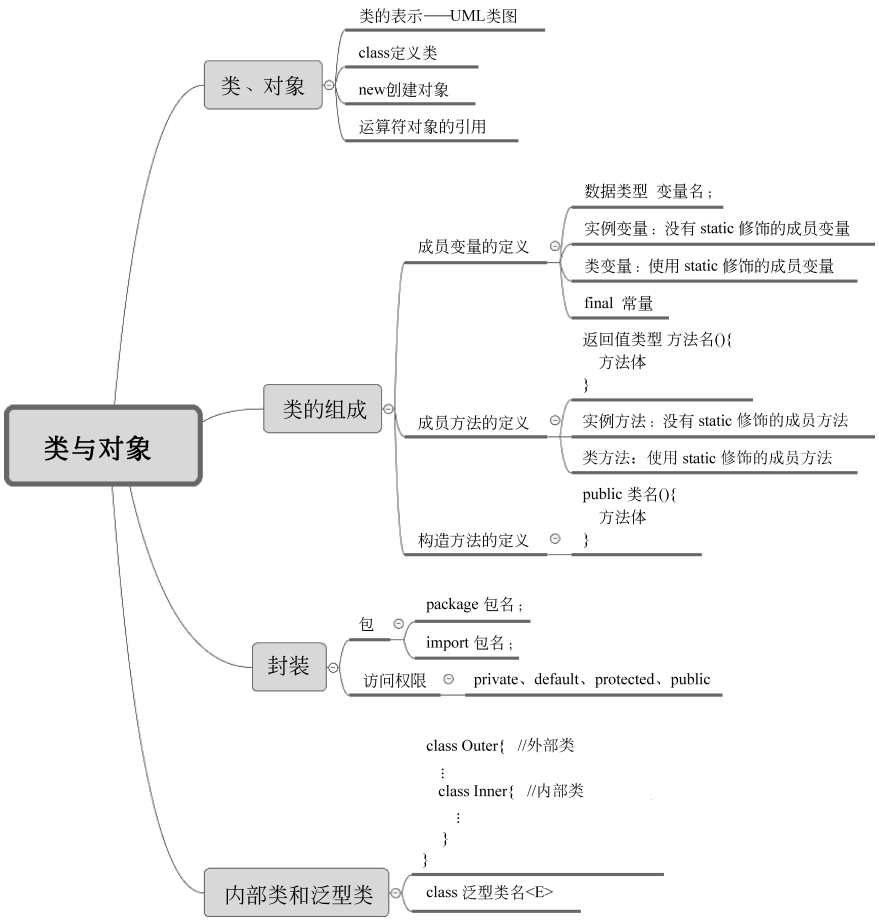
第3章

类 与 对 象

内容导览



学习目标

- 能够根据问题需求运用面向对象思想设计类
- 能够用 Java 语句定义类并运用类创建对象
- 能够运用类和对象以及封装的思想来设计简单的 Java 程序

- 能够识别内部类和泛型类并应用其解决特殊问题

3.1 引　　例

例 3.1　单双号限行制度是为了缓解城市交通压力而催生的一种交通制度。用面向对象的思想来设计一个简单的 Java 程序，查询轿车车牌，输出该车的车牌号为单号还是双号。

说明：单双号的界定是根据车牌号尾数数字来识别，单数为单号，双数为双号。本例不考虑机动车牌号尾数为字母的情况，即默认车牌最后一位为数字。

分析：以面向对象的思想来思考这一问题，不同种类的轿车统称为轿车类。在这个问题中，轿车类有什么共同的静态特征和操作？轿车类的静态特征为车牌（又称为属性），操作（又称为方法）为查询单双号，如图 3-1 所示。

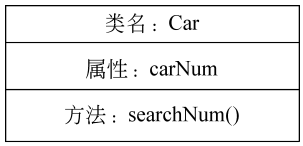


图 3-1　描述轿车类

查询一辆轿车的车牌是单号还是双号，可以通过以下几步来解决：

① 用一个合适的名称（如 Car）来标识要分析的客观实体类（汽车类）。

② 分析客观实体类的共有特征“车牌”，用 carNum 属性对其进行描述。

③ 分析客观实体的共有功能，用 searchNum()方法来实现其功能。

④ 设计轿车类和一个测试类。

⑤ 在测试类中，用轿车类产生一辆轿车对象并计算输出车牌是单号还是双号。

程序代码如下：

```
//文件名 Jpro3_1.java
1  class Car{                                          ← 类名
2      String carNum;                                  ← 成员变量
3      public Car(String num){
4          carNum=num;
5      }
6      void searchNum(){                               ┐
           //charAt()索引范围为从 0 到 length()-1      │
7          char endNum=carNum.charAt(carNum.length()-1);  │
8          int num=Integer.parseInt(String.valueOf(endNum));  ← 成员方法
9          if(num %2==0)                               │
10             System.out.println("该车牌号是双号");   │
11         else                                        │
12             System.out.println("该车牌号是单号");   ┘
13     }
14 }
15 public class Jpro3_1{                               ← 测试类
```

```
16      public static void main(String[] args){ ⟶ main 方法
17          Car myCar=new Car("京 A08L34");
18          myCar.searchNum();
19          myCar.carNum="京 A08L35";
20          myCar.searchNum();
21      }
22  }
```

程序运行结果如下：

```
该车牌号是双号
该车牌号是单号
```

程序分析：在上述 Java 源程序 Jpro3_1.java 中定义了两个类。第一个类为轿车类，类名为 Car，用于实现对轿车实体的共同的属性和方法的封装；第二个类的类名 Jpro3_1 与源程序名相同，该类的主要功能不是用来封装实体，而是用于编写算法对前面的 Car 类的功能进行测试，因此不妨称其为测试类。在测试类中首先定义了一个程序执行的入口方法 main()，在该方法中用轿车类定义了轿车对象，再利用轿车对象调用其相应的方法完成相应的功能。

在解决这个问题时，我们提到了几个概念：类、对象、属性和方法。它们究竟有什么意义？下面将一一介绍。

3.2 认识类和对象

将客观世界中的一个特定种类的实体放在一起并抽取它们身上的共性加以描述，这就得到了软件系统中的类。因此，通常从下面三个方面来描述一个类：

- 有一个名称来唯一标识它所描述的客观实体；
- 有一组属性来描述客观实体的共有特征；
- 有一组方法来实现客观实体的共有行为。

类封装了一类对象的属性和方法，是对象定义的前提。例如，学生类、教师类、课程类、图书类、登录各类网站的用户类等。类是一组具有共同属性和共同行为的对象的统称，是一种复合的数据类型，是组成 Java 程序的基本要素。

3.2.1

3.2.1 认识类

世间万事万物都是客观存在的对象，都可以抽象为包含属性和行为的类。类是具备某些共同特征的实体的集合，是对现实生活中某类对象的抽象，是一种抽象的数据类型。

1. 类的构成

定义类的目的是为了描述一类事物共有的属性和功能，即将数据和对数据的操作封

装在一起，这一过程由类体来实现。类体通常有两种类型的成员：

- 成员变量——通过变量的声明定义来描述类创建的对象的属性。
- 成员方法——通过方法的声明定义来描述类创建的对象的功能。

如何才能做到对类的合理封装呢？这要通过合理地定义类中的成员变量和成员方法来实现。

2. 类的表示——UML 类图

描述同一类对象的属性和行为可采用软件分析或软件设计中的建模语言（UML 类图）进行建模，然后再用 Java 代码实现。UML 类图可以表示类（包括类名、类的属性和操作）和类之间的关系。在 UML 中，类一般表示为一个划分成三格的矩形框，如图 3-2 所示。

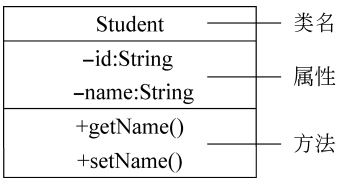


图 3-2　UML 类图示例

在表示类的矩形框中，第一格指定类的名称，即类名。类名应尽量使用领域中的术语，明确且无歧义，以利于开发人员与用户之间的交流。其命名规则与标识符的命名一致。类名的命名规则是类名的第一个字母通常要大写，如果类名是由多个单词连接而成，则每个单词首字母都大写，如 UserDao、LoginController、LoginVo 等。

第二格包含类的属性，用于描述该类对象的共同特点。不同属性具有不同的可见性。常用的可见性有 public、private 和 protected 这三种，在 UML 中分别表示为＋、－和＃。其中，public 可见性表示所有的对象都可以访问；private 表示只有类本身的对象可以访问；protected 表示类本身及其子类的对象可以访问。

第三格表示类的操作，也称为方法，用于修改、查询类的属性或执行其他动作。

UML 建模工具有 Rational Rose、Visio、StarUML 等，具体操作请查阅相关资料。

3. 类的声明

Java 语句中类的定义通常包含两个部分：类声明和类体。其基本格式如下：

```
class 类名{
    类体
}
```

其中，“class 类名”是类的声明部分。class 是关键字，用来定义类。

当定义一个类时，我们可以在“class 类名”前加 public、abstract 和 final 等修饰符对类的特征进行限制；还可以在其后加 extends＜父类名＞和 implements＜接口名列表＞来说明类的继承性。这些内容在后面的章节中将会陆续介绍。

3.2.2　认识对象


3.2.2

对象表示现实世界中某个具体的事物，对象与实体是一一对应的。也就是说，现实世

界中的每个实体都是一个对象，对象是一个具体的概念。

1. 对象的定义

一般情况下，要使用一个类，就必须创建这个类的对象。对象是指一个个的实例，是以类作为“模板”创建的。类是具有共同特性的实体抽象，而对象又是现实世界中实体的表现。类是用来定义对象的模板，当使用一个类创建了一个对象时，也可以说给出了这个类的一个实例。对象是类的实例化，对象(object)和实例(instance)两个词语通常可以互换。当然，实例也可理解为类的具体实现。类和对象的关系是一般与个别的关系，可以比作一张图纸和多幢楼房之间的关系。

对象的创建过程实际上就是类的实例化过程。可使用操作符 new 创建对象，其格式有两种。

```
类名 对象名=new 类名([参数 1,参数 2,…]);
```

例如：

```
Car myCar=new Car("京 A08L34");
```

或者

```
类名 对象名;
对象名=new 类名([参数 1,参数 2,…]);
```

例如：

```
Car myCar;
myCar=new Car("京 A08L34");
```

第一种方式将对象的声明和创建合并在一起，其功能是为对象分配内存空间，然后执行构造方法中的语句为成员变量赋值，最后将所分配存储空间的首地址赋给对象变量。存储空间相当于一个抽屉，数据放在抽屉中，对象变量中所放的是打开这个抽屉的钥匙。其内存模型如图 3-3(b)所示。

第二种方式是先声明对象变量，对象变量声明后，该对象变量还没有引用任何实体，我们称这时的对象为空对象，其内存模型如图 3-3(a)所示。空对象必须再用 new 操作符分配实体后才能使用，其内存模型如图 3-3(b)所示。

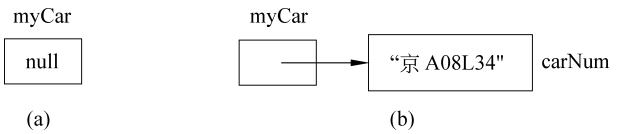


图 3-3　对象的内存模型

使用 new 操作符的结果是返回新创建的对象的一个引用。new 为指定的类创建一个对象时，首先为该对象在内存中分配内存空间，然后以类为模板构造该对象，最后把该对象在内存中的首地址返回给对象名。这样我们就可以像使用一个普通变量一样通过对

象名来使用对象。同时，使用 new 操作符创建对象时，也调用了该类的构造方法实现对象的初始化。

一个类使用 new 运算符可以创建多个不同的对象，这些对象被分配不同的内存空间，因此改变一个对象的内存状态不会影响其他对象的内存状态。例如，我们使用 Car 类创建两个对象 c1 和 c2，其内存模型如图 3-4 所示。

```
Car c1=new Car("京 A08L35");
Car c2=new Car("京 A08L36");
```

图 3-4　多个对象的内存模型

2. main()方法

对象的创建是设计在同一个类中还是应该设计在另一个类中呢？答案是两者都可以，但最好是在另一个类中。这样，没有对象定义的纯粹的类设计部分就可以单独保存在一个文件中，不会影响该类的重复使用。

在例 3.1 中，轿车类和测试类分别放在两个类中来进行设计，这是类重复使用思想的体现。由于 main()方法是每个 Java 应用程序执行的入口方法，因此在 Jpro3_1 类中设计了 main()方法。main()方法的定义格式如下：

```
public static void main(String args[])
```

public 修饰符说明 main()方法可以被所有类访问。static 修饰符表明 main()方法是静态方法。main()方法用于启动 Java 应用程序，因为是用 static 定义的，所以当应用程序启动后，实际上系统中并不存在任何对象，可以直接调用。因此，main()方法的主要工作就是创建启动程序所需的对象。它的返回值类型为 void，即无实际返回值。args[]是形式参数。

3.2.3　对象的使用

3.2.3

一旦创建了对象，就可以使用它编写程序，完成相应的功能。对象的使用主要有以下三种情况。

1. 使用对象的成员变量和成员方法

对象不仅可以操作自己的成员变量来改变状态，而且还可以使用类中的方法，它通过这些方法产生一定的行为。对象通过运算符“.”来引用自己的属性或方法。例如例 3.1 中的第 18、19 行语句：

```
myCar.searchNum();                    //调用方法，查询单双号
myCar.carNum="京 A08L35";             //调用属性，修改车牌号
```

当方法有返回值时，可以将返回值赋给相同类型的变量，也可以直接输出返回值。

2. 对象间的赋值

相同类型的变量可以互相赋值。如果两个对象有相同的值，那么它们就具有相同的实体，即指向同一个内存空间。例如以下语句：

```
Car c1=new Car("京 A08L35");
Car c2=new Car("京 A08L36");
c1=c2;
```

执行赋值语句“c1＝c2;”后，c1 和 c2 引用的实体就一样了，即 c1 和 c2 指向同一个存储空间。c1 原先所引用的存储空间失去了引用对象，变成一块垃圾内存，其内存模型如图 3-5 所示。此时 c1 和 c2 的成员完全相同，其值也相同。

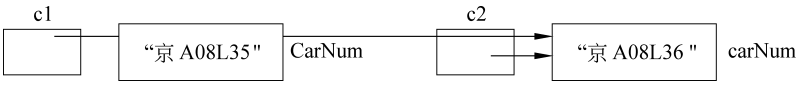


图 3-5　对象赋值后的内存模型

3. 将对象作为方法的参数

对象也可以像变量一样，作为方法的参数使用。

例 3.2　分别定义两个类：学生类和登录服务类，将学生类对象作为登录服务类的成员变量，并且编写测试类进行功能测试。

```
//文件名为 Jpro3_2.java
1  class Student{                                  //定义学生类 Student
2          private String account;
3          private String password;
4          public Student(String account, String password){
5              this.account=account;
6              this.password=password;
7          }
8          public String getAccount(){
9              return account;
10      }
11          public String getPassword(){
12          return password;
13      }
14  }
15  class LoginService{                            //定义登录服务类 LoginService
16      private Student t;                         //定义引用型成员变量
17      public String isLogin(Student t){          //形参为引用型的变量
18          if (t.getAccount()=="admin" && t.getPassword()=="admin"){
19              return "登录成功!";
```

```
20          }
21          return "用户名或密码错误!";
22      }
23  }
24  public class Jpro3_2{                          //定义测试类 Jpro3_2
25      public static void main(String[] args){
26          Student t1=new Student("admin", "admin");
27          LoginService ls=new LoginService();
28          String result=ls.isLogin(t1);      //实参为对象变量
29          System.out.println(result);
30      }
31  }
```

程序运行结果为：

```
登录成功!
```

程序分析：从第 25 行的 main()主方法来阅读程序，第 26、27 行分别创建一个 Student 类的对象 t1、登录服务 LoginService 类的对象 ls，执行的是第 1 行和第 15 行程序；在第 28 行将 t1 作为方法 isLogin()的实参去判断用户登录是否成功，执行的是第 17 行代码。那么实参 t1 传递给形参 t 的是什么呢？是对象变量本身所存储的内容，即所引用对象的内在空间的首地址，而不是所引用的实体的内容。

注意：在第 16 行中，LoginService 类的成员变量 t 的类型为 Student，在第 17 行的 isLogin()登录方法中有这个相应的参数 Student t 来调用其用户名和密码。

3.2.4 垃圾对象的回收

当对象被创建时，会在 Java 虚拟机的堆区中拥有一块内存。在 Java 虚拟机的生命周期中，Java 程序会陆续地创建多个对象，假如所有的对象都永久占有内存，那么内存有可能很快被消耗，引发内存空间不足。因此，必须采取一种措施及时回收那些无用对象占用的内存，以保证内存可以被重复利用。

Java 虚拟机提供了一个系统级的垃圾回收器线程，它负责自动回收那些无用对象所占用的内存，这种内存回收的过程被称为垃圾回收。

在图 3-5 中，c1 和 c2 是 Car 类的两个对象，分别指向两个实体，占用不同的内存空间，如果执行语句

```
c1=c2;
```

则 c1 与 c2 均指向 c2 所引用的内存空间，而 c1 以前指向的内存空间将成为垃圾内存。Java 虚拟机会自动回收这个没用的对象空间。

【练习 3.1】

1.［单选题］一个可以独立运行的 Java 应用程序(　　)。

A. 可以有一个或多个 main()方法　　B. 最多只能有两个 main()方法

C. 可以没有 main()方法　　D. 只能有一个 main()方法

2. [单选题]若要创建 User 类的一个对象 guest,以下书写正确的是(　　)。

A. User guest=new User ();　　B. User guest=new guest();

C. guest=new User();　　D. User guest=A();

3. [单选题]下列说法中能正确地描述类与对象关系的是(　　)。

A. 对象是类的实例化　　B. 对象是抽象的,类通过对象来生成

C. 对象是类的另一个名字　　D. 包含关系

4. [多选题]下列不能正确地定义类的是(　　)。

A. class Person　　B. public class Person

C. new Person　　D. protected class Person

3.3　成员变量和成员方法

Java 中以类来组织程序。类体中所声明的变量被称为成员变量,体现对象的属性。类中的方法称为成员方法,用于对类中声明的变量进行操作,体现对象所具有的行为。

3.3.1　实例变量和类变量

实例变量和类变量都是类的成员变量。成员变量在整个类内有效,其有效范围与在类中书写的先后位置无关。

1. 成员变量

定义成员变量最简单的格式为:

```
类型 变量名 1[,变量名 2,…]
```

变量名前的所有关键字称为该变量的修饰符,变量的类型修饰符是必须有的,它决定该变量在内存中分配的空间的大小。成员变量可以是简单类型,如 byte、int、long、boolean、float、double;也可以是数组、字符串或类等引用类型。

每个类中的成员变量类型的定义要根据具体情况来定,不能一概而论。例如,如果在例 3.1 的 Car 类中增加 speed 属性,该如何定义?

```
class Car{
    String carNum;
    double speed;
}
```

上述代码中关于轿车车牌和速度的定义是否合理呢?

从类的封装性来看，上面的定义并不理想。由于 Car 类中的成员变量的访问权限修饰符为缺省情况，因此在测试类 Jpro3_1 的第 4 行，程序可以对 Car 类的成员变量直接进行操作。

```
1  pubic class Jpro3_1{
2    public static void main(String args[]){
3        Car myCar=new Car("京 A08L34");
4        myCar.speed=60.5;            //调用属性,设置速度
5    }
6  }
```

这就相当于一台电视机除去了机壳，任何人都可以对其里面的器件直接进行操作。没了任何封装，其安全性就受到了威胁。

对于软件系统来说，封装有什么作用呢？在一个包含许多对象的系统中，对象之间以各种方式相互依赖。如果其中一个对象出现了故障，软件工程师不得不修改它，对其他的对象隐藏这个对象的操作意味着只需要修改这个对象而无须改变其他对象。例如上面的 Car 类的定义中，如果将成员变量 speed 的类型由 double 改为 int，那么第 4 行语句将不能通过编译。

要解决上述问题，我们可以在定义成员变量时再加上访问权限修饰符，其格式如下：

```
[访问权限] 类型 变量名 1[,变量名 2,…]
```

Car 类中的成员变量没有使用访问权限修饰符，也就是访问权限修饰符缺省。此时，同一包（具体见 3.4.2 节）中的其他类能对其进行访问。要实现类的成员变量在类的外部不可见，必须使用 private 修饰符对其进行限定。用 private 修饰的成员变量只在本类中有效，因此可以实现数据最严密的封装。

将例 3.1 中关于汽车车牌和速度的定义修改成如下形式：

```
class Car{
    private String carNum;
    private double speed;
}
```

这样，外部类就不能直接访问其实例变量。如果将 Car 类的两个实例变量定义为私有，那么测试类 Jpro3_1 中访问实例变量的语句将无效。例如

```
public class Jpro3_1{
    public static void main(String args[]){
        Car myCar=new Car("京 A08L34");
        myCar.speed=60.5;              //非法
        myCar.carNum="京 A08L35";      //非法
    }
}
```

接下来的问题是：如果外部类无法访问其他类私有的实例变量，那么每个类私有的

实例变量又该如何赋值呢？我们有三种途径来解决这个问题。

- 在定义成员变量时赋初值。
- 在类中定义成员方法给成员变量赋值，一般命名为 setXxx()。
- 利用构造方法给成员变量赋初值。

成员变量定义时如果没有赋值，则其初值是它的默认值。例如，byte、short、int 和 long 类型的默认值为 0，float 类型的默认值为 0.0f，double 类型的默认值为 0.0，boolean 类型的默认值为 false，char 类型的默认值为'\u0000'，引用类型的默认值为 null。但有时我们需要变量具有其他初值，那么可以在定义的同时给变量赋值。例如在定义 Car 类的成员变量时，直接给 speed 赋初值。

```
class Car{
    private double speed=15.5;
}
```

注意以下的写法是错误的。

```
class Car{
    private double speed;
    speed=15.5;                    //非法
}
```

上述程序中的错误反映了这样一个问题：对实例变量的操作应放在方法中进行。当程序执行过程中要改变成员变量的值时，可通过设计相应的 setXxx()方法，在方法体内通过相应的语句来修改。例如

```
public void setSpeed(double speed){          //设置 speed 的值
        this.speed=speed;
    }
```

2. 实例变量

没有用关键字 static 修饰的成员变量称为实例变量。例如在下面的 Student 类中，account 和 password 均为实例变量。

```
class Student{
    private String account;          //实例变量定义
    private String password;         //实例变量定义
}
```

3. 类变量

用关键字 static 修饰的成员变量称为类变量。类变量又称为静态变量。例如在下面的 Student 类中，name 是实例变量，而 number 是类变量。

```
class Student{
    private String account;          //实例变量定义
```

```
    static int number;                    //类变量定义
}
```

具体来说，实例变量和类变量的主要区别为以下三点。

- 内存分配的空间。不同对象的同名实例变量分配不同的内存空间，变量之间的取值互不影响；不同对象的同名类变量分配相同的内存空间，也就是说多个对象共享类变量，改变其中一个对象的类变量的值会影响其他对象中相应的类变量的值。
- 内存分配的时间。当类的字节码文件被加载到内存时，类变量就分配了相应的内存空间；实例变量是当类的对象创建时才会被分配内存。
- 访问方式。实例变量必须用对象名访问；类变量可以用类名访问，也可以用对象名访问。

例 3.3 编写一个学生类，用类变量统计所创建的学生对象的个数。

分析：需要一个变量来存储学生对象个数，多个学生对象要共享一个变量才能记录下总数，因此定义 number 变量为类变量，用于统计学生对象个数。

```
//文件名为 Jpro3_3.java
1   class Student{
2      private String name;
3      static int number=0;                          //定义类变量
4      public Student(String n){                     //定义构造方法
5          name=n;
6          number++;                                 //改变类变量的值
7       }
8   }
9   public class Jpro3_3{
10    public static void main(String args[]){       //用两种方式访问类变量
11        System.out.println("当前学生对象的个数为:"+Student.number);
12          Student s1=new Student("王翰");
13          System.out.println("当前学生对象的个数为:"+s1.number);
14          Student s2=new Student("李媛");
15          System.out.println("当前学生对象的个数为:"+s1.number);
16          System.out.println("当前学生对象的个数为:"+s2.number);
17          System.out.println("当前学生对象的个数为:"+Student.number);
18        }
19  }
```

程序运行结果为：

```
当前学生对象的个数为:0
当前学生对象的个数为:1
当前学生对象的个数为:2
当前学生对象的个数为:2
当前学生对象的个数为:2
```

程序分析：main()方法的第一条语句被执行时，系统首先将 Student 类加载到内存

并为类变量 number 分配内存空间，此时可以用类名 Student 直接引用类变量 number；当执行了第 12 和第 14 行，系统创建了学生对象 s1 和 s2 后，才可以用对象 s1 和 s2 来引用类变量 number，但我们并不提倡使用这种方式，因为其可读性较差。从最后三条语句的输出结果也可看出，s1、s2 和 Student 共享类变量 number。

4. 常量

如果一个类的成员变量前加 final 修饰符，那该成员变量就为常量。常量的名称习惯上用大写字母表示，例如

```
private final double PI=3.14159;
final String SUCCESS="success";
```

常量不占用内存，这意味着在声明常量时必须初始化。对象可以使用常量，但不能更改它的值。

3.3.2 实例方法和类方法

在 Java 中，方法只能作为类的成员，也称为成员方法。类有两种不同类型的成员方法：实例方法和类方法。

1. 成员方法

方法用于操作类所定义的数据并提供对数据访问的代码。大多数情况下，程序都是通过类的方法与其他类的实例进行交互。在类体中，有一些方法的设置是为了实现类的相应功能，例如例 3.1 中的 searchNum()方法用来查询车牌单双号。成员方法往往是类对象的功能体现。

成员方法包括方法声明和方法体。创建成员方法的最简单的格式为：

```
返回值类型 方法名([参数列表]){
    方法体
}
```

第一行为方法声明，大括号中的是方法体。方法体可以包含一个或多个语句，每个方法执行一项任务。每个方法只有一个名称，通过使用这个名称，方法才能被调用。方法名的定义与标识符定义一致，最好能够体现方法的含义，达到“见名知义”的程度。方法名一般用小写字母表示，如例 3.1 中定义的 searchNum()方法。当方法名由多个英文单词组成时，一般第一个单词用小写，后面每个单词的首字母都大写，如 isLogin()、toString()等。

返回值类型是指方法返回值的数据类型。若方法不返回任何值，则返回值类型为关键字 void。除构造方法外，所有的方法都要求有返回值类型。类中可以定义专门操作成员变量的方法，一般命名为 setXxx()和 getXxx()，例如 void setPassword(String password)方法用于设置密码，而 String getPassword()方法用于返回密码的值。

方法名后的参数列表是可选的。列表中的参数称为形式参数，简称为形参。

成员方法也可加访问权限修饰符,用来限定该方法的使用范围。成员方法的访问权限修饰符与成员变量相同,共有四种,其意义也相似。具体内容将在 3.4.2 节中介绍。

2. 实例方法

没有用关键字 static 修饰的成员方法称为实例方法。对上面的 Student 类进行修改,增加两个专门用于操作成员变量的实例方法。

```
1   class Student{
2       private String account;
3       private String password;
4       public String getPassword(){                    //实例方法,获取密码
5           return password;
6       }
7       public void setPassword(String password){       //实例方法,设置密码
8           this.password=password;
9       }
10  }
```

在第 4 行中,getPassword()方法用来获取成员变量 password 的值。在第 7 行中,setPassword()方法用来设置成员变量 password 的值。由于成员变量的访问权限被定义为 private,从 Student 类的外部无法对 password 变量进行访问,因此程序中提供了 setXxx()和 getXxx ()方法用于对该成员变量进行读写操作。如果只设置 getXxx ()方法,那么该变量对外来说是只读的;如果只设置 setXxx()方法,那么该变量对外来说是只写的。

如果希望方法有返回值,则在方法体的最后使用 return xxx;语句,终止方法并返回一个值给该方法的调用者。

3. 类方法

类方法又称为静态方法。用关键字 static 修饰的成员方法称为类方法。在例 3.3 中,若要在 Student 类中定义一个方法来访问静态变量 number,则该方法必须定义为静态方法。

例 3.4 定义静态方法访问静态变量。

```
//文件名为 Jpro3_4.java
1   class Student{
2           private String name;
3           private static int number=0;        //定义类变量
4           public Student(String n){
5               name=n;
6               number++;
7               }
8           public static void print(){         //定义类方法访问类变量
```

```
9               System.out.println("当前学生对象的个数为:"+number);
10          }
11  }
12  public class Jpro3_4{
13      public static void main(String args[]){
14          Student.print();
15          Student s1=new Student("王翰");
16          Student s2=new Student("李媛");
17          Student.print();
18      }
19  }
```

程序运行结果为：

```
当前学生对象的个数为:0
当前学生对象的个数为:2
```

具体来说，实例方法和类方法的主要区别如下。

- 内存分配的时间。当类的字节码文件被加载到内存时，类方法就分配了相应的入口地址；实例方法是当类的对象创建时才会被分配入口地址。
- 访问方式。实例方法必须用对象名访问；类方法一般用类名访问，也可用对象名访问。
- 操作的对象。类方法只能操作类变量，不能操作实例变量；而实例方法既可以操作类变量也可以操作实例变量。
- 另外，实例方法中可以调用实例方法和类方法，而类方法中只能调用类方法，不能调用实例方法。

4. 方法中的参数传递

当方法被调用时，形参被数据或变量替换，这些数据或变量称为实际参数，简称为实参。这种在方法调用时用实参代替形参的形式称为参数传递。

当方法被调用时，如果它有参数，参数变量必须有具体的值。例如构造方法

```
public car(String num){                                    //形参
        carNum=num;
    }

public student(String account, String password){  //形参
        this.account=account;
        this.password=password;
    }
```

其调用语句为：

```
car myCar=new Car("京 A08L34");                    //实参
student t1=new Student("admin", "admin");          //实参
```

在 car()和 student()两个构造方法的声明中设计的参数(String num 以及 String account、String password)均为形式参数。在调用这两个方法时所传递的参数可以是一个常量,如 Car("京 A08L34")中的"京 A08L34";也可以是一个已赋值的基本数据类型的变量,如

```
String a="guest";
String b="123456";
Student s1=new Student(a,b);
```

实参还可以是一个引用类型的对象,如例 3.2 的第 17 行 public String isLogin (Student t) 方法中的 t。调用方法时,实参的值传递给形参。不管参数是何类型,传递时都是按值传递,下面分两种情况进行说明。

(1) 基本数据类型参数的传值

对于基本数据类型的参数,实参数据类型的级别不能高于形参的级别。例如,不能向 int 类型的形参传递一个 float 类型的值,但可以向 double 类型的形参传递一个 float 类型的值。

另外,如果改变形参变量的内容,实参的内容不会跟着变化。

例 3.5 当方法的类型为基本数据类型时,如果改变形参变量的内容,实参的内容不会跟着变化。

```
//文件名为 Jpro3_5.java
1   class Student{
2       public void setPassword(String password){
3           password="abcd";
4           System.out.println("password="+password);
5       }
6   }
7   public class Jpro3_5{
8       public static void main(String args[]){
9           Student t1=new Student();          //创建对象
10          String pwd="1234";
11          t1.setPassword(pwd);               //调用方法,设置 password 的值
12          System.out.println("password="+pwd);
13          System.out.println("pwd="+pwd);
14      }
15  }
```

程序运行结果为:

```
password=abcd
password=1234
pwd=1234
```

程序分析:在第 11 行,当对象 t1 调用方法 setPassword()时,pwd 将 1234 传递给形参 password,password 的值就为 1234,但输出结果执行的是第 4 行代码,即 password=

abcd。

在第 12 行，输出结果是修改后的 password 的值，即 password＝1234。

在第 13 行，pwd 的值并没有改变，仍为 1234，即输出结果为 pwd＝1234。

(2) 引用类型参数的传值

引用类型数据包括对象、数组以及接口等。当方法的参数是引用类型时，实参传递的是对象的引用，而不是对象的内容。

当实参的值传递给形参时，实参和形参对象都指向同一个存储空间。如果改变形参所引用实体的内容，实参所引用实体的内容会跟着变化。

例 3.6 引用类型参数的传值示例。

```
//文件名为 Jpro3_6.java
1  class Point{                //定义圆点类
2     int x;
3     int y;
4     Point(int a,int b){
5       x=a;
6       y=b;
7     }
8   }
9   class Circle{               //定义圆类
10      int r;                  //属性半径
11      Point point;            //属性圆点
12      Circle(int r1,Point p1){
13       r=r1;
14        //p1.x=10;            //改变 p1 的坐标
15        //p1.y=10;
16        point=p1;
17      }
18      void output(){
19          point.x=10;  //改变 point 的坐标
20          point.y=10;
21          System.out.println("圆心的坐标是:"+"("+point.x+","+point.y+")");
22      }
23   }
24  public class Jpro3_6{
25      public static void main(String args[]){
26          Point p=new Point(1,2);
27          System.out.println("点的坐标是:"+"("+p.x+","+p.y+")");
28          Circle c=new Circle(5,p);
29          c.output();
30          System.out.println("点的坐标是:"+"("+p.x+","+p.y+")");
31      }
32  }
```

程序的运行结果为：

```
点的坐标是:(1,2)
圆心的坐标是:(10,10)
点的坐标是:(10,10)
```

程序分析：首先定义了一个坐标为(1,2) 的点对象 p,把 p 作为实参创建一个圆对象 c,将实参 p 的值传递给形参 p1。不管是在构造方法中改变 p1 的坐标,还是在 output() 方法中改变 point 的坐标,p 的坐标都会随之改变。

3.3.3 构造方法

构造方法是一种特殊方法,它的名称必须与它所在类的名称完全相同,并且不返回任何数据类型。Java 程序中的每个类允许定义若干个构造方法,但这些构造方法的参数必须不同。

每个类都有一个默认的构造方法(它没有任何参数)。如果类没有重新定义构造方法,则创建对象时系统自动调用默认的构造方法;否则,创建对象时调用自定义的构造方法。

在例 3.1 的 Car 类中,在定义成员变量的同时给变量赋了初值,这意味着用此类创建的任何轿车对象其初始的车牌号是一样的。如果我们希望每次能得到车牌号不一样的轿车对象,又该如何来设计呢？请看下面的程序：

```
class Car{
    private String carNum;
    public Car(){
        carNum="京 A08L34";
     }
    public Car(String num){
        carNum=num;
      }
    ⋮
}
```

在 Car 类中,增加了两个构造方法,分别用于创建类的对象。那么,在测试类 Jpro3_1 中,就能调用这两个构造方法来创建对象,程序代码如下：

```
public class Jpro3_1{
    public static void main(String args[]){
        Car r1=new Car();
        Car r2=new Car();
        Car r3=new Car("京 A08L38");
        Car r4=new Car("京 A08L39");
        ⋮
    }
}
```

每次调用无参构造方法所创建的轿车对象的车牌号都有一个固定的初值，例如 r1 和 r2 的车牌号均为“京 A08L34”。而每次调用带参构造方法时，只要实参不同，就能构造出不同的轿车对象，例如 r3 的车牌号为“京 A08L38”，而 r4 的车牌号为“京 A08L39”。

3.3.4 关键字 this

this 是 Java 中的关键字，代表本类对象。下面从两个方面介绍它的应用。

1. 使用 this 区分成员变量和局部变量

在方法体中声明的变量以及方法的参数称为局部变量，方法的参数在整个方法内有效，方法内定义的局部变量从它定义的位置之后开始有效。成员变量在整个类内有效。

在一个类中，如果出现局部变量的名称与成员变量的名称相同，则成员变量被隐藏，即这个成员变量在这个方法内暂时失效。例如

```
1  class Car{
2      String carNum;
3      public Car(String carNum){
4          carNum=carNum;              //成员变量被隐藏
5      }
6  }
```

如果希望成员变量 carNum 被成功赋值，必须在其前面加 this。this. carNum 表示当前对象的成员变量 carNum，而不是局部变量 carNum。例如

```
1  class Car{
2      String carNum;
3      public Car(String carNum){
4          this.carNum=carNum;         //成员变量被隐藏
5      }
6  }
```

2. 用 this 调用本类中的其他构造方法

如果在构造方法中用 this 调用本类中的其他构造方法，调用时要放在构造方法的首行。例如

```
class Car{
    private String carNum;
    public Car(){
        this("京 A08L34");          //调用了带参构造方法
    }
    public Car(String num){
        carNum=num;
    }
}
```

注意：实例方法中可以使用 this，类方法中不可使用 this。实例方法中使用 this 来引用的成员表示当前对象的成员，通常情况下省略不写，其含义相同。类方法中不可使用 this，因为类方法可以通过类名直接调用，这时可能还没有创建任何对象。

【练习 3.2】

1. [单选题]下列关于 Java 变量的描述，错误的是(　　)。

A. 在 Java 程序中要使用变量，必须先对其进行声明

B. 类变量可以使用对象名进行调用

C. 变量不可以在其作用域之外使用

D. 成员变量必须写在成员方法之前

2. [单选题]SLOW 是 int 型 public 成员变量，变量值保持为常量 1，可用(　　)语句定义这个变量。

A. public int SLOW=1;　　B. final int SLOW=1;

C. final public int SLOW=1;　　D. public final int SLOW=1;

3. [单选题]以下不属于构造方法特征的是(　　)。

A. 构造方法名与其类名相同　　B. 构造方法有返回值类型

C. 构造方法在创建对象时自动执行　　D. 每一个类可以有多个构造方法

4. [单选题]类 A 有 3 个 int 型成员变量 a、b、c，则(　　)是类 A 的正确构造方法。

A. void A(){a=0; b=0; c=0; }

B. public void A(){a=0; b=0; c=0;}

C. public int A (int x,int y,int z){a=x; b=y; c=z;}

D. public A(int x,int y,int z) {a=x; b=y; c=z;}

5. [单选题]this 关键字的含义是(　　)。

A. 本类　　B. 本类对象　　C. 这个类　　D. 父类对象

6. [单选题]以下(　　)方法是不能编译的？

A.
```
void f(int i) {
    return i;
    }
```

B.
```
void f(int i){
    return 0;
    }
```

C.
```
int f(int i){
    return 0;
    }
```

D.
```
int f(){
    return 0;
    }
```

7. [程序填空]已知圆半径，输出圆的面积和圆的个数。

```
1  class Circle{
2    _______ double PI=3.14;          //定义一个名为 PI 的常量
3    int radius;
4    _______ int num;            //定义静态变量 num,记录创建的 Circle 类对象的个数
5    public Circle(int radius){
6       _______.radius=radius;
7       num++; }
8    double area(){
```