范例 2-51：定义一个无参无返回值的方法。

```
public class TestDemo {
    public static void main(String args[]) {
        printInfo() ;                          //主方法之中直接调用
        printInfo() ;                          //主方法之中直接调用
    }
    public static void printInfo() {           //方法名称
        System.out.println("********************") ;
        System.out.println("*    Hello World.   *") ;
        System.out.println("********************") ;
    }
}
```

程序运行结果：

```
********************
*    Hello World.   *
********************
********************
*    Hello World.   *
********************
```

本程序在主类（TestDemo）中定义了一个 printInfo()方法，而后在主方法中直接调用此方法两次，每次调用都要执行 printInfo()方法定义的输出内容之后返回到被调用处，并且继续向下执行。程序的操作流程如图 2-23 所示。

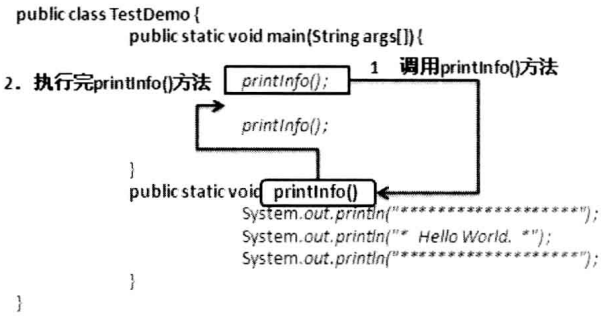


图 2-23 方法执行完后返回到调用处

范例 2-52：定义一个有参无返回值的方法，如将之前打印三角形程序定义为一个方法，每次只需要传入打印的行即可。

```
public class TestDemo {
    public static void main(String args[]) {
        printInfo(3) ;                         //主方法之中直接调用
        printInfo(5) ;                         //主方法之中直接调用
    }
    public static void printInfo(int line) {   //方法名称
        for (int x = 0 ; x < line ; x ++) {    //循环次数，控制行
            for (int y = 0 ; y < line - x ; y ++) {
                System.out.print(" ") ;
            }
            for (int y = 0 ; y <= x ; y ++) {
                System.out.print("* ") ;
            }
            System.out.println() ;
        }
    }
}
```

程序运行结果：

```
  *
 * *
* * *
    *
   * *
  * * *
 * * * *
* * * * *
```

Note

本方法是将之前打印三角形的操作进行了封装，这样用户就可以通过 printInfo()方法执行三角形的输出操作了。

范例 2-53：定义一个有参有返回值的方法。

定义一个方法，用于判断一个数字是奇数还是偶数。很明显，这个方法的返回值类型应该定义为 boolean 比较合适，而且如果一个方法上返回的是 boolean 型数据，则这个方法的名称应该以 **isXxx()**的形式命名。

```
public class TestDemo {
    public static void main(String args[]) {
        if (isType(3)) {                          //isType()返回 boolean 型数据，直接判断
            System.out.println("偶数") ;
        } else {
            System.out.println("奇数") ;
        }
    }
    public static boolean isType(int num) {       //true 表示是偶数，false 表示为奇数
        return num % 2 == 0;                      //是否可以被 2 整除
    }
}
```

程序运行结果：

```
奇数
```

本程序定义的 isType()方法用于判断传入的某一个数字是奇数还是偶数，由于 isType()方法返回的是 boolean 型数据，而 if 语句判断的也是 boolean 型数据，所以可以直接使用方法的返回值作为条件判断。

在定义方法的时候需要额外强调一点，如果一个方法使用了 void 声明，理论上此方法不能够返回数据，但是却可以通过 return 结束调用（即 return 之后的程序不再执行）。

范例 2-54：使用 return 结束方法调用。

```
public class TestDemo {
    public static void main(String args[]) {
        fun(10);                  //调用方法
        fun(30);                  //调用方法
    }
    public static void fun(int num) {
        if (num == 10) {
            return;               //结束方法调用
        }
```

```
        System.out.println("数值：" + num);
    }
}
```

程序运行结果：

数值：30

这一结束的操作和循环控制的 break 与 continue 是一样的，唯一不同的是，此种方式只能用在方法定义上，而且必须保证方法的返回值类型为 void，不过这 3 种操作都离不开 if 语句判断的支持。

2.6.2 方法的重载 重点

方法重载指的是方法名称相同，参数的类型或个数不同，调用的时候将会按照传递的参数类型和个数完成不同的方法体的执行。

范例 2-55：实现方法重载。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println("两个整型相加：" + add(10, 20));
        System.out.println("三个整型相加：" + add(10, 20, 30));
        System.out.println("两个浮点型相加：" + add(10.2, 20.3));
    }
    public static int add(int x, int y) {                    //方法重载
        return x + y;
    }
    public static int add(int x, int y, int z) {             //方法重载
        return x + y + z;
    }
    public static double add(double x, double y) {           //方法重载
        return x + y;
    }
}
```

程序运行结果：

两个整型相加：30

三个整型相加：60

两个浮点型相加：30.5

通过程序可以发现，当一个方法重载之后，会自动根据调用此方法时传入的参数类型或个数的不同，执行不同的方法体，如图 2-24 所示。

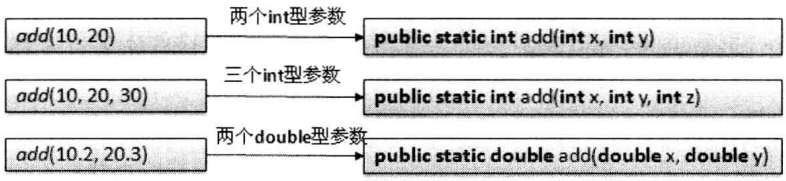


图 2-24 方法重载调用

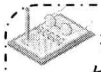
注意：方法重载时只看方法名称、参数类型及个数，而无须关注方法的返回值类型。

范例 2-56：错误的重载。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println("两个整型相加：" + add(10, 20));
        System.out.println("两个浮点型相加：" + add(10, 20));
    }
    public static int add(int x, int y) {    //返回值不同，错误
        return x + y;
    }
    public static double add(int x, int y) { //返回值不同，错误
        return x + y;
    }
}
```

可以发现，这个时候除了方法的返回值类型不一样外，方法的参数类型及个数完全相同，所以这种操作不符合方法重载的定义。

方法重载的时候并没有规定出返回值类型必须统一，即重载的方法返回值类型可以不一样，但是从开发的角度而言，建议所有方法重载之后返回值类型统一。



提示：System.out.println()、System.out.print()也属于方法重载。

在之前一直使用的系统输出操作，实际上也属于一种方法的重载。

范例 2-57：观察如下代码。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println("Hello World") ;    //输出 String
        System.out.println(100) ;              //输出 int
        System.out.println(3000.9) ;           //输出 double
        System.out.println('A') ;              //输出 char
        System.out.println(true) ;             //输出 boolean
    }
}
```

程序运行结果：

```
Hello World
100
3000.9
A
true
```

可以发现，现在 println()方法可以输出各种数据类型，所以此方法为重载方法。

2.6.3 递归调用

递归调用是一种特殊的调用形式，指的是方法自己调用自己的形式，如图 2-25 所示，但是

在进行递归操作的时候必须满足如下的几个条件。

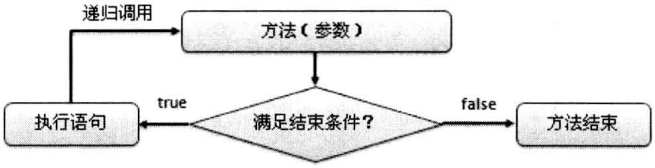


图 2-25 递归调用

Note

☑ 必须有结束条件；

☑ 每次调用的时候都需要改变传递的参数。

范例 2-58：递归操作。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println(add(100));              //执行方法
    }
    public static int add(int num) {
        if (num == 1) {                            //结束条件
            return 1;                              //不再向后继续加了
        }
    return num + add(num - 1);                     //修改参数内容
    }
}
```

程序运行结果：

```
5050
```

本程序使用递归的操作进行了数字的累加操作，并且当传递的参数为 1 时，直接返回一个数字 1。本程序的操作流程简单分析如下：

☑ 第 1 次调用：return 100 + add(99)；

☑ 第 2 次调用：return 100 + 99 + add(98)；

☑ 倒数第 2 次调用：return 100 + 99 + ... + 3 + add(2)；

☑ 最后一次调用：return 100 + 99 + ... + 3 + 2 + 1。

注意：尽量避免使用递归调用。

对于递归操作，现在只要求可以理解这个含义即可，在实际的工作中要尽量少用递归，因为使用不当就可能造成内存溢出。

2.7 习 题 讲 解

学习完了基本的程序逻辑以及方法的定义之后，下面通过 3 道习题来巩固以上的概念，为了帮助大家更好地理解所学习到的概念，以下的 3 道题目都采用两种方式实现。

☑ 方式一：主方法中直接编写程序代码计算；

☑ 方式二：使用递归操作完成。

习题一：要求计算出 1! + 2! + … + 60!的结果。

本程序是进行数字的阶乘累加操作，在操作之前必须首先确定出要使用的数据类型，通过计算发现 60!的结果为 8.320987112741390144276341183223 4e+81，很明显这个数据已经超过了 long 的保存范围，所以本次操作只能使用 double 型数据完成。

实现一：在主方法中直接完成。

```
public class TestDemo {
    public static void main(String args[]) {
        double sum = 0.0 ;                          //保存最终的计算结果
        for (int x = 1 ; x <=60 ; x ++) {
            double temp = 1.0 ;                     //保存每一次阶乘的结果
            for (int y = 1 ; y <= x ;y ++) {
                temp *= y ;                         //计算阶乘
            }
            sum += temp ;                           //保存相加结果
        }
        System.out.println("计算结果：" + sum) ;
    }
}
```

程序运行结果：

```
计算结果：8.46206204346806E81
```

实现二：利用递归完成，需要两次递归。

☑ 递归操作一：负责每一个数据的阶乘操作；

☑ 递归操作二：将每一次的递归操作结果进行相加。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println("计算结果：" + sum(60));
    }
    public static double sum(int num) {             //计算累加
        if (num == 1) {                             //递归结束条件
            return 1;                               //1! = 1
        }
        return mul(num) + sum(num - 1);             //递归
    }
    public static double mul(int num) {             //计算阶乘
        if (num == 1) {
            return 1;
        }
        return num * mul(num - 1);
    }
}
```

程序运行结果：

```
计算结果：8.46206204346806E81
```

习题二：编写一个方法，此方法可以将一个整数变为二进制输出（提示：二进制的计算方法为数字除 2 取余，倒取余数）。

Note

实现一：在主方法中直接完成。

```
public class TestDemo {
    public static void main(String args[]) {
        int num = 18;                          //二进制结果：10010
        String result = "";                    //定义字符串保存
        while (num != 0) {                     //已经没有数值了
            result = (num % 2) + result;       //倒序保存，最早的结果保存在后面
            num = num / 2;                     //改变 num 的内容
        }
        System.out.println(result);
    }
}
```

程序运行结果：

```
10010
```

实现二：通过递归实现。

```
public class TestDemo {
    public static void main(String args[]) {
        toBinary(18);
    }
    public static void toBinary(int num) {
        if (num == 0) {                        //递归结束条件
            return;                            //返回到被调用处
        }
        toBinary(num / 2);                     //递归操作
        System.out.print(num % 2);             //输出结果，先递归（向下计算）后输出
    }
}
```

程序运行结果：

```
10010
```

习题三：有 5 个人坐在一起，问第 5 个人多少岁？答：比第 4 个人大 2 岁，问第 4 个人多少岁的时候，比第 3 个人大 2 岁，问第 3 个人多少岁的时候，比第二个人大 2 岁，问第一个人多少岁的时候，第一个人说自己是 8 岁，那么第 5 个人的岁数是？

实现一：在主方法中直接完成。

```
public class TestDemo {
    public static void main(String args[]) {
        int age = 8;                           //第一个人的岁数
        for (int x = 0; x < 4; x++) {          //4 次循环，5 个人
            age += 2;
        }
        System.out.println(age);
    }
}
```

程序运行结果：

```
16
```

实现二：通过递归。

```
public class TestDemo {
    public static void main(String args[]) {
        System.out.println(age(5)) ;
    }
    public static int age(int num) {          //人数
        if (num == 1) {
            return 8 ;                        //第一个人
        }
        return 2 + age(num - 1) ;             //计算累加年龄
    }
}
```

程序运行结果：

16

提问：代码是写在主方法里好，还是写在自定义方法里好？

通过以上 3 道习题的两种不同做法，觉得将代码写在主方法里就可以解决问题了，为什么还要写第二种方式呢？在开发中该选择什么样的方式编写会更好？

回答：主方法可以理解为一个客户端，代码越少越好。

如果从实际的开发而言，本书推荐使用第二种方式（但并不推荐递归），即将所有的操作都交给不同的方法完成，主方法只编写很少的调用即可，养成这样的习惯，可以更好地帮助用户理解正规开发模式。

2.8 本章小结

1．Java 的数据类型可分为两种：基本数据类型和引用数据类型。

2．UNICODE 为每个字符制定了一个唯一的数值，在任何语言、平台、程序都可以安心地使用。

3．布尔（boolean）类型的变量，只有 true（真）和 false（假）两个值。

4．数据类型的转换可分为下列两种：自动类型转换与强制类型转换。

5．算术运算符的成员有加法运算符、减法运算符、乘法运算符、除法运算符、余数运算符。

6．if 语句可依据判断的结果来决定程序的流程。

7．递增与递减运算符有着相当大的便利性，善用它们可提高程序的简洁程度。

8．括号是用来处理表达式的优先级的，也是 Java 的运算符。

9．需要重复执行某项功能时，循环就是最好的选择。可以根据程序的需求与习惯，选择使用 Java 所提供的 for、while 及 do...while 循环来完成。

10．break 语句可以让程序强制逃离循环。当程序运行到 break 语句时，会离开循环，继续执行循环外的下一个语句。如果 break 语句出现在嵌套循环中的内层循环，则 break 语句只会逃离当前层循环。

11．continue 语句可以强制程序跳到循环的起始处，当程序运行到 continue 语句时，会停止

运行剩余的循环主体，而返回到循环的开始处继续运行。

12．选择结构包括了 if、if...else 及 switch 语句，语句中加上了选择结构后，就像是十字路口，根据不同的选择，程序的运行会有不同的方向与结果。

13．方法是一段可重复调用的代码段，在本章中因为方法可以由主方法直接调用，所以要加入 public static 关键字修饰。

14．方法的重载：方法名称相同，参数的类型或个数不同，则此方法被称为重载。

Note

2.9 实践与练习

2.9.1 简答题

1．请解释常量与变量的区别。

2．解释方法重载的概念，并举例说明。

2.9.2 编程题

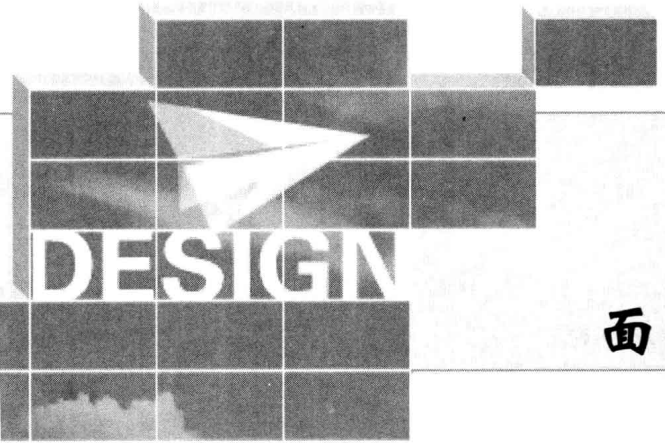
1．打印出 100~1000 范围内的所有“水仙花数”，所谓“水仙花数”是指一个 3 位数，其各位数字立方和等于该数本身。例如，153 是一个“水仙花数”，因为 153=1 的三次方＋5 的三次方＋3 的三次方。

2．通过代码完成两个整数内容的交换。

3．判断某数能否被 3、5、7 同时整除。

4．编写程序，分别利用 while、do...while 和 for 循环求出 100~200 的累加和。

第 2 部分



面向对象

- 面向对象
- 异常的捕获及处理
- 包及访问控制权限
- Java 新特性

第3章

面向对象

通过本章的学习，可以达到以下目标：

- ☑ 掌握面向对象的主要特点；
- ☑ 掌握类与对象的定义格式及内存分配；
- ☑ 掌握封装型的主要特点及实现要求；
- ☑ 掌握构造方法的定义形式、主要特点和使用限制；
- ☑ 掌握简单 Java 类的开发原则；
- ☑ 掌握数组的基本使用及内存分配；
- ☑ 掌握 static 关键字的使用；
- ☑ 掌握内部类以及匿名内部类的特点及使用；
- ☑ 掌握继承性的主要作用、实现、使用限制；
- ☑ 掌握方法覆写的操作；
- ☑ 掌握 final 关键字的使用；
- ☑ 掌握抽象类和接口的定义、使用、常见设计模式；
- ☑ 掌握 Object 类的主要特点及实际应用；
- ☑ 掌握数据表和简单 Java 类的关系映射操作；
- ☑ 理解常见数据结构的开发。

学习完 Java 基础语法之后，下面最为重要的内容就是面向对象，可以说面向对象是整个 Java 的灵魂所在。本章将开始为读者详细地讲解面向对象的各个核心概念，包括每一概念的实际应用。

3.1 面向对象简介

Note

面向对象是现在最为流行的程序设计方法之一，现代的程序开发几乎都是以面向对象为基础。但是在面向对象设计之前，广泛采用的是面向过程，面向过程只是针对自己来解决问题。面向过程的操作是以程序的基本功能实现为主，实现之后就完成了，也不考虑修改的可能性；面向对象，更多的是要进行子模块化的设计，每一个模块都需要单独存在，并且可以被重复利用，所以面向对象的开发更像是一个具备标准的开发模式。

提示：面向过程与面向对象的区别。

考虑到读者暂时还没有掌握面向对象的概念，所以本书先使用一些较为直白的方式帮助读者理解面向过程与面向对象的区别。例如，现在要想制造一把手枪，则可以有两种做法。

- ☑ 做法一（面向过程）：将制造手枪所需的材料准备好，由个人负责指定手枪的标准，如枪杆长度、扳机设置等，但是这样做出来的手枪完全只是为一把手枪的规格服务，如果某个零件（如扳机坏了）需要更换的时候，那么就必须首先弄清楚这把手枪的制造规格，才可以进行生产，所以这种做法没有标准化和通用性；
- ☑ 做法二（面向对象）：首先由一个设计人员设计出手枪中各个零件的标准，并且不同的零件交给不同的制造部门，各个部门按照标准生产，最后统一由一个部门进行组装，这样即使某一个零件坏掉了，也可以轻易地进行维修，这样的设计更加具备通用性与标准模块化设计要求。

对于面向对象的程序设计有 3 个主要的特性：封装性、继承性和多态性。下面为读者简单介绍一下这 3 种特性，在本书后面的内容中会对此三个方面进行完整的阐述。

1. 封装性

封装是面向对象的方法所应遵循的一个重要原则，它有两个含义：一是指把对象的属性和行为看成一个密不可分的整体，将这两者“封装”在一个不可分割的独立单位（即对象）中。另一层含义指“信息隐蔽”，把不需要让外界知道的信息隐藏起来，有些对象的属性及行为允许外界用户知道或使用，但不允许更改，而另一些属性或行为，则不允许外界知晓，或只允许使用对象的功能，而尽可能隐蔽对象的功能实现细节。

封装机制在程序设计中表现为，把描述对象属性的变量及实现对象功能的方法合在一起，定义为一个程序单位，并保证外界不能任意更改其内部的属性值，也不能任意调动其内部的功能方法。

封装机制的另一个特点是，为封装在一个整体内的变量及方法规定了不同级别的可见性或访问权限。

2. 继承性

继承是面向对象方法中的重要概念，并且是提高软件开发效率的重要手段。

首先拥有反映事物一般特性的类，然后在其基础上派生出反映特殊事物的类。如已有的汽车的类，该类中描述了汽车的普遍属性和行为，进一步再产生轿车的类，轿车的类是继承于汽

车类，轿车类不但拥有汽车类的全部属性和行为，还增加轿车特有的属性和行为。

在 Java 程序设计中，已有的类可以是 Java 开发环境所提供的一批最基本的程序——类库。用户开发的程序类是继承这些已有的类。这样，现在类所描述过的属性及行为，即已定义的变量和方法，在继承产生的类中完全可以使用。被继承的类称为父类或超类，而经继承产生的类称为子类或派生类。根据继承机制，派生类继承了超类的所有成员，并相应地增加了自己的一些新成员。

Note

面向对象程序设计中的继承机制，大大增强了程序代码的可复用性，提高了软件的开发效率，降低了程序产生错误的可能性，也为程序的修改扩充提供了便利。

若一个子类只允许继承一个父类，称为单继承；若允许继承多个父类，称为多继承。目前许多面向对象程序设计语言不支持多继承，而 Java 语言通过接口（interface）的方式来弥补由于 Java 不支持多继承而带来的子类不能享用多个父类的成员的缺憾。

3. 多态性

多态是面向对象程序设计的又一个重要特征，是允许程序中出现重名现象。Java 语言中含有方法重载与对象多态两种形式的多态。

- ☑ 方法重载：在一个类中，允许多个方法使用同一个名字，但方法的参数不同，完成的功能也不同。
- ☑ 对象多态：子类对象可以与父类对象进行相互的转换，而且根据其使用的子类的不同完成的功能也不同。

多态的特性使程序的抽象程度和简捷程度更高，有助于程序设计人员对程序的分组协同开发。

3.2 类 与 对 象 重点

3.2.1 类与对象的基本概念

在面向对象中类和对象是最基本、最重要的组成单元，那么什么叫类呢？类实际上是表示一个客观世界某类群体的一些基本特征的抽象，属于抽象的概念集合。而对象则是表示一个个具体的事物，如张三同学、李四账户、王五的汽车，这些都是可以使用的事物，那么就可以理解为对象，所以对象表示的是一个个独立的个体。

例如，在现实生活中，人就可以表示为一个类，因为人本身属于一种广义的概念，并不是一个具体的个体描述。而某一个具体的人，如张三同学，就可以称为对象，可以通过各种信息完整地描述这个具体的人，如这个人的姓名、年龄、性别等信息，那么这些信息在面向对象的概念中就称为属性，当然人是可以吃饭、睡觉的，那么这些人的行为在类中就称为方法。也就是说如果要使用一个类，就一定有产生对象，每个对象之间是靠各个属性的不同来进行区分的，而每个对象所具备的操作就是类中规定好的方法，类与对象的关系如图 3-1 所示。

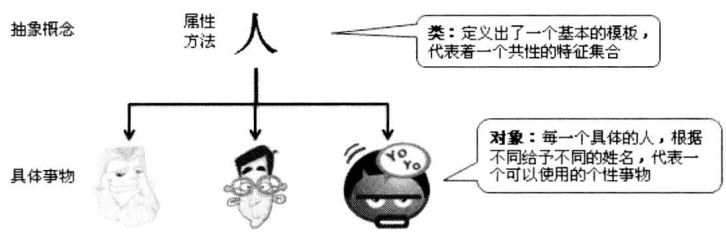


图 3-1 类与对象的概念

提示：类与对象的简单理解。

在面向对象中有这样一句话可以很好地解释类与对象的区别：“类是对象的模板，而对象是类的实例”，即对象所具备的所有行为都是由类来定义的，按照这种方式理解，在开发中应该先定义出类的结构，之后再通过对象来使用这个类。

提示：类与对象的另一种解释。

关于类与对象，初学者在理解上是存在一定难度的，笔者在此再为各位读者做一个简单的比喻，读者应该都很清楚，如果要想生产出汽车，则首先一定要设计出一个汽车的设计图纸（如图 3-2 所示），之后按照此图纸规定的结构生产汽车。这样生产出的汽车结构和功能都是一样的，但是每辆车的具体内容，如各个汽车的颜色、是否有天窗等都会存在一些差异。

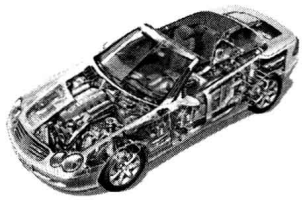


图 3-2　汽车设计图纸

在这个实例中，汽车设计图纸实际上就是规定出了汽车应该有的基本组成，包括外型、内部结构、发动机等信息的定义，那么这个图纸就可以称为一个类，显然只有图纸是无法使用的；而通过这个模型产生出的一辆辆的具体汽车是可以被用户使用的，所以就可以称其为对象。

3.2.2　类与对象的定义

从之前的概念中可以了解到，类是由属性和方法组成的。属性中定义类一个个的具体信息，实际上一个属性就是一个变量，而方法是一些操作的行为，但是在程序设计中，定义类也是要按照具体的语法要求完成的，如果要定义类需要使用 class 关键字，类的定义语法如下。

提示：属性也可以称为成员。

类中的属性实际上相当于一个个的变量，有时候也称之为 Field（成员）。

格式 3-1：类的定义格式。

```
class 类名称{
    数据类型  属性（变量）;
    ....                                    声明成员变量（属性）

    public 返回值的数据类型 方法名称(参数 1,参数 2...){
        程序语句 ;
        [return 表达式;]                                    定义方法的内容
    }
}
```

范例 3-1：定义一个 Person 类。

```
class Person {                  //类名称首字母大写
    String name;                //定义类成员（属性）
    int age;                    //定义类成员（属性）
    public void tell() {        //定义类方法，没有 static
```