第5章 创建类的成员属性和方法

主要知识点

Java 语言的基本组成；

运算符与表达式；

程序控制结构；

Java 程序的编程规范；

类成员方法的创建方法。

学习目标

能根据 Java 语言的基本语法和程序结构声明类的成员方法，从而定义完整的类。

在第 4 章中学习了类的定义方法，能够对类的属性进行声明。一个完整的类是由若干属性和方法构成。本章将在学习 Java 语法的基础上，学会定义类的成员方法，从而能够编写简单的应用程序，通过“仿 QQ 聊天软件”项目的分析，掌握类的定义与用法。

5.1 Java 语言的基本组成

Java 语言主要由 5 种元素组成：标识符、关键字、文字、运算符和分隔符。各元素有着不同的语法含义和组成规则，它们互相配合，共同完成 Java 语言的语意表达。

5.1.1 分隔符

分隔符用于将一条语句分成若干部分，便于系统识别，让编译程序确认代码在何处分隔。Java 语言的分隔符有三种：空白符、注释语句、普通分隔符。

1. 空白符

在 Java 程序中，换行符和回车键均表示一行的结束，都是典型的空白符，空格键和 Tab 制表健也是空白符。为了增加程序可读性，Java 语句的成分之间可以插入任何多个空白符，在编译时，系统自动忽略多余的空白符。

2. 注释语句

注释语句是为了提高程序的可读性而附加的内容，编译程序对注释部分既不显示也不执行，有没有注释部分不会影响程序的执行。Java 提供了三种形式的注释：

(1) // 一行的注释内容

以//开始,最后以按回车键结束,表示从//到本行结束的所有字符均为注释内容。

(2) /* 一行或多行的注释内容 */

从/* 到 */ 间的所有字符(可能包括几行内容)都作为注释内容。

以上两种注释可用于程序的任何位置。

(3) /** 文档注释内容 */

当这类注释出现在任何声明之前时将会作特殊处理,它们不能再用在代码的任何地方。表示被括起来的正文部分,应该作为声明项目的描述,而被包含在自动产生的文档中。

编写程序时,除了加入适当的空白符和注释内容外,还要尽可能使用层次缩进格式,使同一层语句的起始列号位置相同,层次分明,便于阅读和维护,形成良好的编程习惯。

3. 普通分隔符

普通分隔符用于区分程序中的各种基本成分,但它在程序中有确切的意义,不可忽略。包括 4 种:

- 花括号({ }),用来定义复合语句、类体、方法体以及进行数组的初始化。
- 分号(;),表示一条语句的结束。
- 逗号(,),用来分隔变量说明和方法的参数。
- 冒号(:),说明语句标号。

例 5-1　分隔符的用法。

```
/** 程序名:MyClass.java
功能:简单程序使用举例
开发日期:2022 年 3 月 10 日
*/
public class MyClass{                                    //这里定义一个类
   public static void main(String args[]){               //定义程序的主方法
        int int1,int2;                                   //声明 2 个整型变量
        /* 从这里开始书写方法的内容 */
        …
   }                                                     //main()方法结束
}                                                        //MyClass 类结束
```

程序说明:对于应用程序,必须有一个主方法 main(),是程序执行的入口,其中的参数格式是 String args[],表示含有 String 字符串类型的数组参数 args,args 是数组名,是用户自定义标识符,可以改成其他名称。以上参数也可以写成 String[] args 的形式。main()方法的返回值是 void,表示无返回值,是公有的(public),是类的静态成员(static)。

5.1.2　关键字

关键字用来表示特定的意义,也叫保留字,由系统本身使用,不能用作标识符。Java 的关键字共有 48 个,所有的关键字都是小写字母,如表 5-1 所示。

表 5-1 Java 关键字

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	null	package
private	protected	public	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		

5.2 运算符与表达式

运算符指明对操作数所进行的运算。按操作数的数目来分,可以有单目运算符(如++、--),双目运算符(如+、>)和三目运算符(如?:),它们分别对应于一个、两个和三个操作数。对于单目运算符来说,可以有前缀表达式(如++i)和后缀表达式(如 i++),对于双目运算符来说则采用中缀表达式(如 a+b)。按照运算符功能来分,基本的运算符有下面几类:

- 算术运算符(+,-,*,/,%,++,--)
- 关系运算符(>,<,>=,<=,==,!=)
- 布尔逻辑运算符(!,&&,||)
- 位运算符(>>,<<,>>>,&,|,^,~)
- 赋值运算符(=,及其扩展赋值运算符如+=)
- 条件运算符(?:)
- 其他,包括分量运算符(.)、下标运算符[]、实例运算符 instanceof、内存分配运算符 new、强制类型转换运算符(类型)、方法调用运算符()等。

5.2.1 算术运算符

算术运算符作用于整型或浮点型数据,完成算术运算。

1. 双目算术运算符

双目算术运算符包括+、-、*、/、%(取模)5 种运算符。

Java 对加运算符进行了扩展,使它能够进行字符串的连接,如"abc"+"de",得到串"abcde"。取模运算符%的操作数可以是浮点数,如 37.2%10=7.2。

2. 单目算术运算符

单目算术运算符如表 5-2 所示(op 表示操作数)。

表 5-2　单目运算符

运　算　符	用　　法	描　　述
＋	＋ op	正值
－	－ op	负值
＋＋	＋＋ op,op ＋＋	加 1
－－	－－ op,op －－	减 1

i＋＋与＋＋i 的区别：i＋＋在使用 i 之后，使 i 的值加 1，因此执行完 i＋＋后，整个表达式的结果仍为 i，而 i 的值变为了 i＋1。＋＋i 在使用 i 之前，使 i 的值加 1，因此执行完＋＋i 后，整个表达式和 i 的值均为 i＋1。例如，初值 a＝1，b＝1，执行 a＝b＋＋后 a＝1，b＝2，而执行 a＝＋＋b 后，a＝2，b＝2。

5.2.2　关系运算符

关系运算符用来比较两个值的大小关系，返回布尔类型的值 true 或 false。关系运算符都是双目运算符，包括＞、＞＝、＜、＜＝、＝＝、！＝、＜＞共 7 个。

在 Java 中，任何数据类型的数据（包括基本类型和组合类型）都可以通过＝＝或！＝来比较是否相等，结果返回 true 或 false。关系运算符常与布尔逻辑运算符一起使用，作为流控制语句的判断条件。

5.2.3　逻辑运算符

逻辑运算符包括 &&（逻辑与）、||（逻辑或）、！（逻辑非），逻辑表达式的结果是一个布尔值 true 或 false。

对于布尔逻辑运算，先求出运算符左边的表达式的值，对于或运算，如果为 true，则整个表达式的结果为 true，不必再对运算符右边的表达式再进行运算；同样，对于与运算，如果左边表达式的值为 false，则不必再对右边的表达式求值，整个表达式的结果为 false，这种逻辑运算又称为逻辑短路与和逻辑短路或。

例 5-2　逻辑运算符的应用。

```
public class Test502{
  public static void main(String args[]){
    int a = 25,b = 3;
    boolean d = a < b;                              //d = false
    System.out.println("a < b = " + d);
    int e = 3;
    if(e!= 0 && a/e > 5)
      System.out.println("a/e = " + a/e);
    int f = 0;
    if(f!= 0 && a/f > 5)                            //注意此语句中被 0 除
      System.out.println("a/f = " + a/f);
    else
      System.out.println("f = " + f);
  }
}
```

说明：第二个 if 语句在运行时不会发生除 0 溢出的错误，因为 f!=0 已经为 false，所以就不需要再对 a/e 进行运算。

5.2.4 赋值运算符

赋值运算符“=”把一个数据赋给一个变量，在赋值运算符两侧的类型不一致的情况下，如果左侧变量的数据类型的级别高，则右侧的数据被转化为与左侧相同的数据类型，然后赋给左侧变量；否则，需要使用强制类型转换运算符，如：

```
byte b = 100;                                    //自动转换
int i = b; int j = 100;
byte b = (byte)a;                                //强制类型转换
```

在赋值符“=”前加上其他运算符，即构成扩展赋值运算符，例如：a+=3 等价于 a=a+3，用扩展赋值运算符可表达为：变量 运算符=表达式，如 x*=5。

5.2.5 条件运算符

条件运算符 ?：是三目运算符，一般形式为：

```
expression? statement1: statement2
```

其中，表达式 expression 的值应为一个布尔值，如果该值为 true，则执行语句 statement1；否则执行语句 statement2，而且语句 statement1 和 statement2 需要返回相同的数据类型，且该类型不能是 void。例如：

```
money = score>=60 ? 100:0;
```

表示：如果 score>=60，则 money=100，否则 money=0。

如果要通过测试某个表达式的值来选择两个表达式中的一个进行计算时，用条件运算符来实现是一种简捷的方法，实现了 if-else 条件语句的功能。如求 a 和 b 的较大值，表达式是

```
max = a>b?a:b
```

5.2.6 表达式

表达式是变量、常量、运算符、方法调用的序列，它执行这些元素指定的计算并返回某个值，如 a+b，c+d 等都是表达式。表达式用于计算并对变量赋值，以及作为程序控制的条件。表达式的值由表达式中的各个元素来决定，可以是简单类型，也可以是复合类型。

在对一个表达式进行运算时，要按运算符的优先顺序从高向低进行，同级的运算符则按从左到右的方向进行，通过加()可以提高运算符的优先级，因为小括号的优先级最高。因此在表达式中，可以用括号()显式地标明运算次序，括号中的表达式首先被计算。适当地使用括号可以使表达式的结构清晰。例如：a>=b&&c<d||e==f 可以用括号显式地写成((a<=b)&&(c<d))||(e==f)，这样就清楚地表明了运算次序，使程序的可读性加强。

技能训练 2　创建类的成员属性

一、目的

(1) 掌握面向对象的基本概念；
(2) 掌握定义类与创建对象实例的方法；
(3) 掌握类属性的定义和使用；
(4) 培养良好的编码习惯和编程风格。

二、内容

1. 任务描述

用户 User 在“仿 QQ 聊天软件”中是一个非常重要的类，服务器使用该类管理用户的相关信息，这里假设服务器只需要维护用户的账号 id、密码 password 两个属性。类的属性见表 5-T-1。

表 5-T-1　User 类属性定义

类名	User
属性	id、password

2. 实训步骤

(1) 打开 Eclipse 开发工具，新建一个 Java Project，项目名称为 Ch05Train_1，项目的其他设置采用默认设置。注意当前项目文件的保存路径。

(2) 在新建的 Ch05Train_1 项目中添加一个名为 User 的类，包名称为空。

打开 User.java 文件，编写如下代码：

```
public class User {
    public String id;       //账号
    public String pwd;      //密码
    /*
     * 无参构造方法
     */
    public User() {
    }
}
```

(3) 为了测试 User 类，向 Ch05Train_1 项目中再添加一个类名为 TestUser 的测试类，在 TestUser 类的 main()方法中实现以下功能。

① 用 User 类创建一个实例 user，并在控制台输出 user 对象的所有属性。

② 修改密码为“234”，输出修改后的密码。

```
/**
 * 对 User 类的属性和方法进行测试
 */
public class TestUser {
    public static void main(String[] args) {
        //使用带两个参数的构造方法创建对象 user,账号为 2020001,密码为 123
        User user = new User();
        user.id = "2020001";
        user.pwd = "123";
        System.out.println("账号:" + user.id);            //输出账号
        System.out.println("密码:" + user.pwd);           //输出密码
        user.pwd = "234";                                 //修改密码
        System.out.println("修改后的密码:" + user.pwd); //输出修改后的密码
    }
}
```

(4) 当没有编译错误时,切换到 TestUser.java 窗口,单击 Run 菜单下的 Run 菜单项运行 TestUser 类,在 Eclipse 控制台中显示程序的运行结果如图 5-T-1 所示。

3. 任务拓展

完善用户 User 类的定义。给 User 类增加一个属性 String petName,用于保存用户的昵称,修改用户 2020001 的昵称为"张三",并输出用户 2020001 的昵称,输出内容如图 5-T-2。

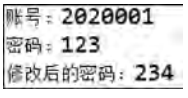

图 5-T-1 运行结果

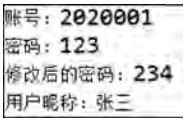

图 5-T-2 运行结果

4. 思考题

在前面程序的基础上,思考如下的问题:

(1) 类成员属性的命名规则是什么?

(2) boolean 类型 get 方法的习惯写法与其他数据类型有何不同?

(3) 对于 User 类中的成员变量 id、password、petName 的修饰符都为 public,修改成 private 程序可以运行吗? 如果不行,需要做哪些修改,使用 private 有什么好处吗?

三、独立实践

在"仿 QQ 聊天软件"中,一个用户 User 可以有多个朋友 Friend,请实现这两个类之间的一对多关系。

5.3 控制结构

Java 程序通过流程控制语句来执行程序,完成一定的任务。被控制的程序段可以是单

一的一条语句，也可以是用大括号{ }括起来的一个复合语句。Java 中的控制结构，包括：

- 分支语句：if-else，break，switch，return。
- 循环语句：while，do-while，for，continue。
- 异常处理语句：try-catch-finally，throw。

5.3.1　分支语句

分支语句提供了一种控制机制，使程序在执行时可以跳过某些语句，而转去执行特定的语句。

1. 条件语句 if-else

if-else 语句根据判定条件的真假来执行两种操作中的一种，格式为：

```
if(boolean - expression)
  statement1;
[else
statement2;]
```

- 布尔表达式 boolean-expression 是任意一个返回布尔型数据的表达式。
- 每个单一的语句后都必须有分号。
- 语句 statement1、statement2 可以为复合语句，这时要用大括号({ })括起来，建议对单一的语句也用大括号括起，这样程序的可读性强，而且有利于程序的扩充，大括号外面不加分号。
- else 子句是任选的。
- 若布尔表达式的值为 true，则程序执行 statement1，否则执行 statement2。
- if-else 语句的一种特殊形式为嵌套语句，即：

```
if(expression1){
  statement1
  }else if (expression2){
           statement2
         }…
       }else if (expressionM){
              statementM
            }else {
       statementN
}
```

else 子句不能单独作为语句使用，必须和 if 配对使用，它总是与离它最近的 if 配对，可以通过使用大括号{ }来改变配对关系。

例 5-3　判断某一年是否为闰年，闰年的 2 月是 29 天，平年是 28 天。闰年的条件是符合二者之一：① 用 4 位数表示的年份能被 4 整除，但不能被 100 整除；② 能被 400 整除。

```
public class LeapYear{
  public static void main(String args[]){
    int year = 1989;                                    //方法 1
```

```
    if((year % 4 == 0 && year % 100!= 0) || (year % 400 == 0))
      System.out.println(year + "是闰年");
    else
      System.out.println(year + "不是闰年");
    year = 2000;                                          //方法 2
    boolean leap;
    if(year % 4!= 0) leap = false;
    else if(year % 100!= 0)
         leap = true;
    else if(year % 400!= 0)
      leap = false;
    else
      leap = true;
    if(leap == true)
      System.out.println(year + "是闰年");
    else
      System.out.println(year + "不是闰年");
    year = 2050;                                          //方法 3
    if(year % 4 == 0){
      if(year % 100 == 0){
      if(year % 400 == 0)
        leap = true;
      else
        leap = false;
      }else
        leap = false;
      }else
        leap = false;
    if(leap == true)
      System.out.println(year + "是闰年");
    else
      System.out.println(year + "不是闰年");
  }
}
```

运行结果为：

```
1989 不是闰年
2000 是闰年
2050 不是闰年
```

说明：方法 1 用一个逻辑表达式包含了所有的闰年条件；方法 2 使用了 if-else 语句的特殊形式；方法 3 则通过使用大括号({})对 if-else 进行匹配来实现闰年的判断。

2. 多分支选择语句 switch

switch 语句根据表达式的值来执行多个操作中的一个，一般格式如下：

```
switch (expression){
  case value1 : statement1;
  break;
```

```
  case value2 : statement2;
  break;
  …
  case valueN : statemendN;
  break;
  [default : defaultStatement; ]
}
```

说明：

- 表达式 expression 可以返回任一简单类型的值(如整型、实型、字符型)，多分支语句把表达式返回的值与每个 case 子句中的值相比。如果匹配成功，则执行该 case 子句后的语句序列。
- case 子句中的值 value1 必须是常量，而且所有 case 子句中的值应是不同的。
- default 子句是任选的。当表达式的值与任一 case 子句中的值都不匹配时，程序执行 default 后面的语句；如果表达式的值与任一 case 子句中的值都不匹配且没有 default 子句，则程序不作任何操作，而是直接跳出 switch 语句。
- break 语句用来在执行完一个 case 分支后，使程序跳出 switch 语句，即终止 switch 语句的执行。因为 case 子句只是起到一个标号的作用，用来查找匹配的入口并从此处开始执行，对后面的 case 子句不再进行匹配，而是直接执行其后的语句序列，因此应该在每个 case 分支后，要用 break 来终止后面的 case 分支语句的执行。在一些特殊情况下，多个不同的 case 值要执行一组相同的操作，这时可以不用 break。
- case 分支中包括多个执行语句时，可以不用大括号{ }括起。

switch 语句的功能可以用 if-else 来实现，但在某些情况下，使用 switch 语句更简练，可读性强，而且程序的执行效率提高。

例 5-4　根据考试成绩的等级打印出百分制分数段。

```
public class Test504{
  public static void main(String args[]){
    System.out.println("\n**** first situation **** ");
    char grade = 'C';                                    //normal use
    switch(grade){
      case 'A' : System.out.println(grade + " is 85～100");
                 break;
      case 'B' : System.out.println(grade + " is 70～84");
                 break;
      case 'C' : System.out.println(grade + " is 60～69");
                 break;
      case 'D' : System.out.println(grade + " is < 60");
                 break;
      default : System.out.println("input error");
    }
    System.out.println("\n**** second situation **** ");
    grade = 'A';                                         //成绩赋值为 A
    switch(grade){
       case 'A' : System.out.println(grade + " is 85～100");
       case 'B' : System.out.println(grade + " is 70～84");
```

```
            case 'C' : System.out.println(grade + " is 60～69");
            case 'D' : System.out.println(grade + " is < 60");
            default : System.out.println("input error");
        }
        System.out.println("\n **** third situation **** ");
        grade = 'B';                                    //several case with same operation
        switch(grade){
            case 'A' :
            case 'B' :
            case 'C' : System.out.println(grade + " is >= 60");
            break;
            case 'D' : System.out.println(grade + " is < 60");
            break;
            default : System.out.println("input error");
        }
    }
}
```

从该例可以看到 break 语句的作用,如果子句中缺少了 break 语句,输出结果会出错。

3. break 语句

在 switch 语中,break 语句用来终止 switch 语句的执行,使程序从 switch 语句后的第一个语句开始执行。可以为每个代码块加一个括号,一个代码块通常是用大括号({ })括起来的一段代码。加标号的格式如下:

```
BlockLabel: { codeBlock }
```

break 语句的第二种使用情况就是跳出它所指定的块,并从紧跟该块的第一条语句处执行。其格式为:

```
break BlockLabel;
```

即用 break 来实现程序流程的跳转,不过应该尽量避免使用这种方式。

4. 返回语句 return

return 语句从当前方法中退出,返回到调用该方法的语句处,并从紧跟该语句的下一条语句继续程序的执行。返回语句有两种格式:

格式 1:

```
return expression;
```

用于返回一个值给调用该方法的语句,返回值的数据类型必须和方法声明中的返回值类型一致。可以使用强制类型转换来使类型一致。

格式 2:

```
return;
```

当方法说明中用 void 声明返回类型为空时,应使用这种格式,它不返回任何值。return 语句通常用在一个方法体的最后,以退出该方法并返回一个值。Java 中,单独的 return 语

句用在一个方法体的中间时，会产生编译错误，因为这时会有一些语句执行不到。但可以通过把 return 语句嵌入某些语句(如 if-else)来使程序在未执行完方法中的所有语句时退出，例如：

```
int method(int num) {
return num;                                          //本句会产生编译错误
if (num > 0)
return num;
…     // 本句能否执行,取决于 num 的值
}
```

5.3.2　循环语句

循环语句的作用是反复执行一段代码，直到满足终止循环的条件为止，一个循环一般应包括四部分内容。

- 初始化部分(initialization)：设置循环的一些初始条件，如计数器清零等。
- 循环体部分(body)：反复循环的一段代码，可以是单一的一条语句，也可以是复合语句。
- 迭代部分(iteration)：这是在当前循环结束，下一次循环开始前执行的语句，常常用来使计数器变量加 1 或减 1。
- 终止部分(termination)：通常是一个布尔表达式，每一次循环都要对该表达式求值，以验证是否满足循环终止条件。

Java 提供的循环语句有 while、do-while 和 for 三种语句，下面分别介绍。

1. while 语句

while 语句实现“当型”循环，一般格式为：

```
[initialization]       //初始化
while (termination){   //循环条件
  body;                //循环体
[iteration;]           //迭代,改变循环变量的值
}
```

布尔表达式(termination)表示循环条件，值为 true 时，循环执行大括号中的语句。while 语句首先判断循环条件，当条件成立时，才执行循环体中的语句，如果一开始条件就不成立，可能循环体一次都不会执行，这是“当型”循环的特点。

2. do-while 语句

do-while 语句实现“直到型”循环，一般格式为：

```
[initialization]
do {
  body;
[iteration;]
} while (termination); //
```

do-while 语句首先执行循环体，然后判断循环条件，若结果为 true，则循环执行大括号

中的语句,直到布尔表达式的结果为 false。与 while 语句不同的是,do-while 语句的循环体至少执行一次,这是“直到型”循环的特点。

3. for 语句

for 语句实现固定次数的循环,一般格式为:

```
for (initialization; termination; iteration){
  body;
}
```

- for 语句执行时,首先执行初始化操作,然后判断循环条件是否满足,如果满足,则执行循环体中的语句,最后执行迭代部分。完成一次循环后,重新判断终止条件。
- 可以在 for 语句的初始化部分声明一个变量,它的作用域为整个 for 语句。
- for 语句通常用来执行循环次数确定的情况(如对数组元素进行操作),也可以根据循环结束条件执行循环次数不确定的情况。
- 在初始化部分和迭代部分可以使用逗号语句来进行多个操作。逗号语句是用逗号分隔的语句序列。例如:for(i=0,j=10; i<j; i++,j--){…}。
- 初始化、终止以及迭代部分都可以为空语句,但分号不能省,三者均为空时,相当于一个无限循环(死循环)。

4. continue 语句

continue 语句用来结束本次循环,跳过循环体中下面尚未执行的语句,接着进行终止条件的判断,以决定是否继续循环。对于 for 语句,在进行终止条件的判断前,还要先执行迭代语句。它的格式为:

```
continue;
```

也可以用 continue 跳转到括号指明的外层循环中,格式为:

```
continue outerLable;
```

例 5-5 用 while、do-while 和 for 语句实现累计求和。

```
public class Test505{
  public static void main(String args[]){
    System.out.println("\n**** while statement ****");
    int n = 10,sum = 0;                    //初始化
    while(n > 0){                          //循环条件
      sum += n;                            //循环体
      n -- ;                               //迭代
    }
    System.out.println("sum is " + sum);
    System.out.println("\n**** do_while statement ****");
    n = 0;                                 //初始化
    sum = 0;
    do{
      sum += n;                            //循环体
```

```
      n++;                          //迭代
    }while(n<=10);                  //循环条件
    System.out.println("sum is "+sum);
    System.out.println("\n**** for statement ****");
    sum=0;
    for(int i=1; i<=10; i++){       //初始化,循环条件,迭代
      sum+=i;                       //循环体
    }
    System.out.println("sum is "+sum);
  }
}
```

运行结果为：

```
**** while statement ****
sum is 55
**** do_while statement ****
sum is 55
**** for statement ****
sum is 55
```

可以从中来比较这三种循环语句，从而在不同的场合选择合适的语句。

5.3.3　Java 编程规范

养成良好的编程风格是程序员应具备的基本素质，运用 Java 编程也要遵守 Java 编程规范，这对于读懂别人的程序和让别人理解自己的程序都十分重要。

1. 一般原则

- 尽量使用完整的英文单词描述符。
- 采用适用于相关领域的术语。
- 采用大小写混用，可读性更好。
- 避免使用相似或相同的名字，或者仅仅是大小写不同的名字。
- 少用下画线（除静态常量等）。

2. 具体要求

- 包（Package）：包名采用完整的英文描述符，都由小写字母组成。
- 类（Class）：类名采用完整的英文描述符，所有单词的第一个字母均大写，如 CustomerName，SavingsAccountBank。
- 接口（Interface）：接口名采用完整的英文描述符说明接口封装，所有单词的第一个字母大写。习惯上，接口名字后面加上后缀 able、ible 或者 er，但这不是必需的。如 Contactable、Prompter。
- 组件（Component）：使用完整的英文描述来说明组件的用途，末端应接上组件类型，如 okButton、customerList、fileMenu。
- 异常（Exception）：通常采用字母 e 表示异常的实例，这是个特例，表示单词

Exception 的第一个字母,易于记忆。

- 变量、属性、方法:采用完整的英文描述,第一个单词首字母小写,后面所有单词的首字母大写。如 firstName、lastName。
- 获取成员函数:被访问字段名的前面加上前缀 get,如 getFirstName()、getLastName()。
- 布尔型获取成员函数:所有的布尔型获取函数必须用单词 is 做前缀,表示"是不是……"这样一个意义,如 isPersistent()、isString()。
- 设置成员函数:被访问字段名的前面加上前缀 set 表示设置,如 setFirstName()、setLastName()、setWarpSpeed()。
- 普通成员函数:采用完整的英文描述说明成员函数功能,第一个单词尽可能采用动词,第一个字母小写,类似于变量名,如 openFile()、addAccount()。
- 静态常量(static final):全部采用大写字母,单词之间用下画线分隔,如 PI、MIN_BALANCE、DEFAULT_DATE。
- 循环变量:用于循环语句中控制循环次数,通常用 i、j、k 或者 counter 表示。

5.4 数组

数组是有序数据的集合,数组中的每个元素具有相同的类型,数组名用下标来区分数组中的元素位置,数组可分为一维数组和多维数组。

5.4.1 一维数组

1. 一维数组的定义

一维数组的定义格式为:

```
type arrayName[];
```

或者

```
type [] arrayName;
```

其中类型 type 可以为 Java 中任意的数据类型,包括简单类型和复合类型(也可以是数组),数组名 arrayName 为一个合法的标识符,[]指明该变量是一个数组类型变量。例如:

```
int intArray[];
int [] intArray;
```

声明了一个整型数组,数组中的每个元素为整型数据。Java 在数组的定义中并不为数组元素分配内存,因此[]中不能指出数组中元素个数(数组长度),不允许访问这个数组的任何元素。必须为它分配存储空间后才能访问它的元素,这时要用到 new 命令,其格式是:

```
arrayName = new type[arraySize];
```

arraySize 指明数组的长度。如:"intArray=new int[10];"为一个整型数组分配 10 个

int 型整数所占据的内存空间，其下标分量为 0～9。

通常，这两部分可以合在一起，用一条语句完成，格式是：

```
type arrayName = new type[arraySize];
```

例如：

```
int intArray = new int[3];
```

2. 一维数组元素的引用

定义了一个数组，并用运算符 new 为它分配了内存空间后，就可以引用数组中的每一个元素了。数组元素的引用方式为：

```
arrayName[index]
```

其中，index 为数组下标，它可以为整型常数或表达式。如 a[8]、b[i](i 为整型)、c[2 * i]等。下标从 0 开始，一直到数组的长度减 1。对于上面例子中的 intArray 数来说，它有 10 个元素，分别从 intArray[0]到 intArray[9]，但没有 intArray[10]。

Java 对数组元素要进行越界检查以保证安全性。同时，每个数组都有一个属性 length 指明它的长度(元素个数)，例如：intArray. length 表示数组 intArray 的长度。

3. 一维数组的初始化

对数组元素可以按照上述的例子进行赋值，也可以在定义数组的同时初始化。例如：

```
int a[] = {1,2,3,4,5};
```

但“int a[5]={1,2,3,4,5};”是非法的，系统自动统计数据个数，不要用户确定。

用逗号(,)分隔数组的各个元素，系统自动为数组分配一定的存储空间。

例 5-6　从小到大冒泡法排序数组，冒泡排序是对相邻的两个元素进行比较，并把小的元素交换到前面。

```
public class Test506{
  public static void main(String args[]){
    int i,j;
    int intArray[] = {30,1, - 9,70,25};          //数组初始化
    int l = intArray.length;                     //数组长度赋值给变量 l
    for(i = 0;i < l - 1;i++)
      for(j = i + 1;j < l;j++)
        if(intArray[i]> intArray[j]){            //以下 3 行用于交换位置
          int t = intArray[i];
          intArray[i] = intArray[j];
          intArray[j] = t;
        }
      for(i = 0;i < l;i++)
        System.out.println(intArray[i] + "");
  }
}
```

5.4.2 多维数组

多维数组可以看作是数组的数组。例如二维数组的每个元素又是一个一维数组。下面以二维数组为例来进行说明,多维数组的使用与此类似。

1. 二维数组的定义

二维数组实际上就是一个表,矩阵或者行列式,每个元素的位置均包括行号(第一维)和列号(第二维),其定义格式为:

```
type arrayName[][];
```

例如:

```
int intArray[][];
```

与一维数组一样,这时对数组元素也没有分配内存空间,使用运算符 new 生成一个数组对象后,系统才会分配内存,才能访问每个元素。

对多维数组来说,分配内存空间有下面几种方法:

(1) 直接为每一维分配空间,如:

```
int a[][] = new int[2][3];
```

(2) 从最高维开始,分别为每一维分配空间,如:

```
int a[][] = new int[2][];
a[0] = new int[3];
a[1] = new int[3];
```

2. 二维数组元素的引用

对二维数组中每个元素,引用格式为:

```
arrayName[index1][index2]
```

其中,index1、index2 均为下标,可以是整型常数或表达式,如 a[2][3]等。同样,每一维的下标都是从 0 开始编号。

3. 二维数组的初始化

有两种方式初始化:直接对每个元素进行赋值、在定义数组的同时进行初始化。如:

```
int a[][] = {{2,3},{1,5},{3,4}};
```

定义了一个 3×2 的数组,并对每个元素赋值。

例 5-7 二维数组举例——矩阵的乘法运算。

两个矩阵 $\boldsymbol{A}_{m\times p}$、$\boldsymbol{B}_{p\times n}$ 相乘得到 $\boldsymbol{C}_{m\times n}$,每个元素 $C_{ij}=\sum_{k=1}^{p}a_{ik}b_{kj}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{ip}b_{pj}(i=1\sim m,j=1\sim n)$

```
public class Test507{
  public static void main(String args[]){
    int i,j,k;
    int a[][] = new int[2][3];                                    //数组 a 赋值
    int b[][] = {{1,5,2,8},{5,9,10,-3},{2,7,-5,-18}};             //数组 b 赋值
    int c[][] = new int[2][4];
    for(i = 0;i < 2;i++)
      for(j = 0;j < 3;j++)
        a[i][j] = (i + 1) * (j + 2);
      for(i = 0;i < 2;i++){
        for(j = 0;j < 4;j++){
          c[i][j] = 0;
          for(k = 0;k < 3;k++)
          c[i][j] += a[i][k] * b[k][j];
        }
      }
      System.out.println("\n*** MatrixA *** ");
      for(i = 0;i < 2;i++){
        for(j = 0;j < 3;j++)
          System.out.print(a[i][j] + "");
          System.out.println();
        }
        System.out.println("\n*** MatrixB *** ");
        for(i = 0;i < 3;i++){
          for(j = 0;j < 4;j++)
            System.out.print(b[i][j] + "");
          System.out.println();
        }
        System.out.println("\n*** MatrixC *** ");
        for(i = 0;i < 2;i++){
          for(j = 0;j < 4;j++)
            System.out.print(c[i][j] + "");
            System.out.println();
        }
    }
}
```

以上介绍的数组都要求固定大小，即声明时就需要确定分量的个数。有时可能会出现这种情况：在声明时无法确定大小，运行过程中大小可变，这时要使用类 ArrayList，它由系统包 java.util 提供，用法是，首先预定义一个初始大小（通常为 10），当向数组中添加元素时，其长度会自动增加。ArrayList 类的用法，读者可以查阅 JDK 帮助文档。

5.5　成员方法的声明

类的方法，也叫作成员函数，用来规定类属性上的操作，实现类的内部功能；同时，它也是类与外界联系的渠道，即外界通过调用类的方法来实现信息交互。

5.5.1　方法的声明

声明类的方法的格式是：

```
[修饰符] 返回值类型 方法名(形式参数列表) [throws 异常名列表]
{   方法体;
    局部变量声明;
    语句序列;
}
```

方法名是一个标识符,由用户定义,在同一个类中允许有同名方法,但其参数(包括参数名、类型、个数、顺序)不能完全相同,而且允许有同名的成员属性名和方法名,只是不建议这样使用。

方法的修饰符很多,包括访问控制修饰符、静态修饰符 static、抽象方法修饰符 abstract、最终方法 final 等。

访问控制修饰符和静态修饰符 static 的作用与成员属性相同。

用 final 声明的方法称为最终方法,它不能被子类覆盖,即不能在子类中修改或者重新定义,但可以被子类继承,用于保护一些重要的方法不被修改。

用 abstract 修饰的方法称为抽象方法,这种方法只有方法头的声明,没有方法体,它不能实现。只有被重写时,加上方法体后,才能产生对象。抽象方法只能出现在抽象类中,含有抽象方法的类称为抽象类,抽象类中还可以含有其他非抽象类。一个抽象类可以定义一个统一的编程接口,使其子类表现出共同的状态和行为,但各自的细节不同。子类共有的行为由抽象类中的抽象方法来约束,而子类行为的具体细节由通过抽象方法的覆盖来实现。这种机制可以增加编程的灵活性。

throws 子句用于异常处理,这些内容在第 11 章介绍。

在方法声明中,必须返回一个类型,如果只是执行一个操作,没有确定的类型,则返回的是 void。

在方法中声明的变量只在本方法中有用,离开本方法不可以再引用,而成员属性的作用范围是整个类,因此它与成员属性是不同的。

下面的类 Student 声明了 5 个属性和 2 个方法:

```
public class Student{
  private int age;                    //声明私有变量 age
  private String name;                //声明私有变量 name
  private char sex;                   //声明私有变量 sex
  public final int GRADE = 2;         //声明公有的 final 变量 GRADE,即常量
  public static int counter = 0;      //声明公有的 static 型变量 counter
  public void speekEnglish(){
    System.out.println("大家好,我在说英语!");
  }
  public void walk(){
    System.out.println("我在散步!");
  }
}
```

下面的 MyScore 类声明了一个属性 result 和一个求和的方法 sum,在方法中定义了形式参数 a 和 b 以及局部变量 x,注意它们的作用范围。

```
public class MyScore{
```

```
    public int result;                    //result 的作用范围是整个类
  public void sum(int a,int b){           //a 和 b 的作用范围是方法内
    int x;                                //x 是局部变量,作用范围是方法内
    x = a + b;
    result = x;
    System.out.println("x = " + x);
   }
}
```

5.5.2　方法的覆盖与重载

Java 是通过方法的覆盖和重载来实现多态的。类层次结构中,如果子类中的一个方法与父类中的方法有相同的方法名并具有相同数量和类型的参数列表,则称子类中的方法覆盖了父类中的方法。通过子类引用覆盖方法时,总是引用子类定义的方法,而父类中定义的方法被隐藏。

在子类中,若要使用父类中被隐藏的方法,可以使用 Super 关键字。

1. 方法的覆盖

例 5-8　方法覆盖示例。

```
class SuperClass{                         //父类声明
  public void printA(){
    System.out.println("父类打印函数");
  }
}
class SubClass extends SuperClass{        //子类声明
  public void printA(){
    System.out.println("子类打印函数");
  }
}
public class OverrideDemo {
  public static void main(String[] args) {
    SuperClass s1 = new SubClass();       //s1 是子类的实例
    s1.printA();
  }
}
```

以上程序的输出结果:

```
子类打印函数
```

在 OverrideDemo 类中,语句"SuperClass s1＝new SubClass();"创建了一个类型为 SuperClass 的对象,而且 SubClass 对象被赋给 SuperClass 类型的引用 s1。语句:s1.printA()将调用 SubClass 的 printA()方法,因为 s1 引用的对象类型为 SubClass。

父类的引用变量可以引用子类对象。Java 用这一事实来解决在运行期间对覆盖方法的调用。过程为:当一个覆盖方法通过父类引用被调用时,Java 根据被引用对象的类型来决定执行哪个版本的方法。因此,如果父类包含一个被子类覆盖的方法,那么通过父类引用

变量引用不同子对象时,就会执行该方法的不同版本。

覆盖方法允许 Java 支持运行时多态性。多态性是面向对象的编程本质,因为它允许通用类指定方法,这些方法对该类的派生类都是公用的。同时该方法允许子类定义这些方法中某些或全部实现。覆盖方法是 Java 实现多态性的一种方式。

2. 方法的重载

方法的重载是 Java 实现面向对象的多态性机制的另一种方式。在同一个类中两个或两个以上的方法可以有相同的名字,只要它们的参数声明不同即可,这种情况称为方法重载。Java 用参数的类型和数量来确定实际调用的重载方法的版本。因此每个重载的方法的参数的类型或数量必须是不同的。虽然每个重载方法可以有不同的返回类型,但返回类型并不足以区分所使用的是哪个方法。当 Java 调用一个重载方法时,参数与调用参数匹配的方法被执行。

例 5-9 方法重载示例,程序名为 OverLoadDemo.java。

```
class Calculation{
  public void add(int a,int b){
    int c = a + b;
    System.out.println("两个整数相加得:" + c);
  }
  public void add(float a,float b){
    float c = a + b;
    System.out.println("两个浮点数相加得:" + c);
  }
  public void add(String a,String b){
    String c = a + b;
    System.out.println("两个字符串相加得:" + c);
  }
}
public class OverLoadDemo {
  public static void main(String[] args) {
    Calculation c = new Calculation();
    c.add(10,20);
    c.add(21.5f,32.3f);
    c.add("早上","好");
  }
}
```

以上程序的输出结果是:

```
两个整数相加得: 30
两个浮点数相加得: 53.8
两个字符串相加得: 早上好
```

在以上程序中,add()方法被重载。其中有三个方法具有相同的名称 add(),但具有不同的参数,分别实现两个整数、浮点数相加和两个字符串连接。在实际调用时,具体调用哪个 add 方法取决于传递的参数与哪个方法相匹配。

前面所有例题的变量赋值都是在编程时确定的,如果希望变量的值在程序运行过程中

从键盘动态输入，则需要使用 Scanner 类。Scanner 类由系统包 java.util 提供，即 java.util.Scanner，它是一个用于扫描输入文本的类，用于在程序运行时接收用户从键盘输入的信息，可以接收字符串和 char、int、double、boolean 等多种类型的数据。

Scanner 类提供的常用方法如下。

- next()：读取下一个值。
- nextInt()：读取下一个整数。
- nextBoolean()：读取下一个布尔型数据。
- nextByte()：读取下一个字节型数据。
- nextShort()：读取下一个短整型数据。
- nextLong()：读取下一个长整型数据。
- nextDouble()：读取下一个双精度型数据。
- nextFloat()：读取下一个浮点型数据。
- nextLine()：读取下一行数据。

要使用 Scanner 命令，则必须在程序首部添加一条命令：import java.util.Scanner；然后在类中创建 Scanner 对象，如：Scanner s = new Scanner(System.in)；这样才可以调用 Scanner 的方法从键盘读取数据。

例 5-10　输入若干学生成绩，用－1 结束输入，输出平均成绩。

```
import java.util.Scanner;                    //引入类,也可以是 import java.util.*;
public class ExamScore{
      public static void main(String args[]){
      float cj, pj, zh = 0, n = 0;
      Scanner s = new Scanner(System.in);    //生成 Scanner 的实例 s
      cj = s.nextFloat();                    //调用 nextFloat 方法从键盘读取输入的值赋给 cj
      while (cj!= -1){
      zh += cj;
      n += 1;
      cj = s.nextFloat();                    //继续读取下一个成绩
      }
   if (n!= 0) {
   pj = zh/n;
      System.out.println("平均成绩是:" + pj);
      }
   else System.out.println("您没有输入成绩");
   }
}
```

程序运行时，系统会等待用户输入数据，直到输入－1 为止才显示结果。用户输入数据时，可以每输入一个数据按一下回车键，也可以一行输入多个数据，中间用空格分开。

特别提示：从本章实例可以看出，编程是运用数据模型来解决日常工作中的问题，比如闰年 2 月份的天数，这就需要大家关注生活。同时，一个问题可能有多种解决方案，也有多种编程方法，应具有选择最优方案的知识储备，编写出来的软件用户才会喜欢。

技能训练 3　创建类的成员方法

一、目的

(1) 掌握面向对象的基本概念；
(2) 掌握定义类与创建对象实例的方法；
(3) 掌握类属性的定义和使用；
(4) 掌握 get 和 set 方法的定义使用；
(5) 掌握类成员方法的定义和使用；
(6) 培养良好的编码习惯和编程风格。

二、内容

1. 任务描述

好友在“仿 QQ 聊天软件”中也是一个非常重要的类，它记录了用户的好友信息，通常一个用户的好友信息至少包括用户的账号 id、好友姓名 friendName、好友的账号 friendId 等信息。请编写一个好友类 Friend，实现相关属性和方法。属性与方法见表 5-T-2。

表 5-T-2　Friend 类定义

类名	Friend
属性	id，friendName，friendId
方法	getId，setId(String id)，getFriendId()，setFriendId(String friendId)

2. 实训步骤

(1) 打开 Eclipse 开发工具，新建一个 Java Project，项目名称为 Ch05Train_2，项目的其他设置采用默认设置。注意当前项目文件的保存路径。

(2) 在新建的 Ch05Train_2 项目中添加一个名为 Friend 的类，包名称为空。

打开 Friend.java 文件，编写如下代码：

```
/*
 * 用户好友信息
 */
public class Friend {
    private String id;              //用户账号
    private String friendId;        //好友账号
    public Friend() {
        id = "2020001";
        friendId = "2020002"; }
    /*
     * 获得用户账号
     */
```

```
    public String getId() {
        return id; }
    /*
     * 设置用户账号
     */
    public void setId(String id) {
        this.id = id; }
    /*
     * 获得好友账号
     */
    public String getFriendId() {
        return friendId; }
    /*
     * 设置好友账号
     */
    public void setFriendId(String friendId) {
        this.friendId = friendId; }
}
```

(3) 为了测试 Friend 类，向 Ch05Train_2 项目中再添加一个类名为 TestFriend 的测试类，在 TestFriend 类的 main()方法中实现以下功能：

① 用 Friend 类创建一个实例 friend，在控制台输出 friend 对象的所有属性(用户账号、好友账号)；

② 修改好友账号为 201901，输出修改后的 friend 对象的所有属性。

```
/*
 * 测试用户好友信息
 */
public class TestFriend {
    public static void main(String[] args){
        Friend friend = new Friend();                                  //创建好友对象
        System.out.println("用户账号:" + friend.getId());              //输出用户账号
        System.out.println("好友账号:" + friend.getFriendId());}       //好友账号
}
```

(4) 当没有编译错误时，切换到 TestFriend.java 窗口，单击 Run 菜单下的 Run 菜单项运行 TestFriend 类，在 Eclipse 控制台中显示程序的运行结果如图 5-T-3 所示。

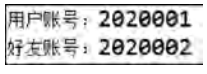

图 5-T-3　运行结果

3. 任务拓展

(1) 完善账户 Friend 类的定义，增加属性：好友姓名 friendName，并为属性 friendName 添加相应的 get 和 set 方法；

(2) 在 Friend 类中增加验证好友的方法：boolean checkFriend(String id, String friendId)，如果 id 值与 friendId 相等(用户账号不能与好友账号相同)，返回 false；否则返回 true。编写代码进行测试。

4. 思考题

在前面程序的基础上,思考如下的问题:

(1) 类成员方法的命名规则是什么?

(2) 修饰类的成员方法时,说一说使用 public、protected 和 private 的不同。

三、独立实践

(1) 计算器具有加 add、减 sub、乘 mul、除 div 四种功能,编写一个计算器类 Calculator,并编写测试类 TestCalculator 类,对计算器 Calculator 进行测试;

(2) 编写日期类 MyDate,具有年 year、月 month、日 day 三个属性和输出日期 string getDate()、判断是否是闰年 boolean isLeapYear()两个方法。请实现 MyDate 类并测试。

本章习题

1. 简答题

(1) Java 提供了哪些注释语句,功能有什么不同?

(2) 识别下面标识符,哪些是合法的,哪些是非法的。

Ply_1,$32,java,myMothod,While,your-list,class,ourFriendGroup_$110,长度,7st

(3) 程序有哪三种控制结构?

(4) Java 中提供了哪些循环控制语句?

(5) 数组有什么特点,数组的声明和初始化方法与简单变量有什么不同?

(6) Java 编码规范有哪些?

(7) 什么是方法的覆盖? 什么是方法的重载?

2. 编程与操作题

(1) 编写一个类,其方法是从 10 个数中求出最大值、最小值及平均值。

(2) 编写一个类,其方法是编程求 $n!$,设 $n=8$。

(3) 编写一个类,其方法是:根据考试成绩的等级打印出分数段,优秀为 90 分以上,良好为 80～90 分,中等为 70～79 分,及格为 60～69 分,60 分以下为不及格,要求采用 switch 语句。

(4) 编写一个类,其方法是:判断一个数是不是回文。回文是一种从前向后读和从后向前读都一样的文字或者数字,如 12321、569878965、abcba。

(5) 编写一个类,其方法是:将数组中值按从大到小排列输出。

(6) 编写一个类,其方法是:编程输出杨辉三角形的前 8 行(杨辉三角形的构成特点是:每一行两边的数均为 1,中间的数等于它肩上的两个数之和,这是我国南宋数学家杨辉 1261 年在所著的《详解九章算术》一书中提出的,而欧洲的帕斯卡(1623—1662)在 1654 年才发现

这一规律，比杨辉迟了 393 年，表明中华民族灿烂文化源远流长）。

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
```