第 3 章

类的方法

类的方法是对一类对象的共同行为的抽象，对一种行为的描述往往需要比较复杂的流程控制，例如判断和选择，以及对同一代码段的多次重复执行。当程序运行过程中遇到异常情况时，需要通过异常处理机制，将程序流程导向到专门的错误处理模块。本章将详细介绍方法的流程控制和异常处理机制。

3.1 方法的控制流程

Java 程序通过控制语句来控制方法的执行流程。类的方法中语句可以是简单语句，也可以是复合语句。简单语句是以分号“;”结尾的单语句，复合语句是一对大括号{和}括起来的语句组，也称为“块”，最后没有分号。如：

```
{  }

{
    BankAccount bankAccount1=new BankAccount("LinLi", 500);
    System.out.println(bankAccount1);
}
```

第一个块是个空块，其中不含任何语句。第二个块包含两条语句。

与过程化程序设计语言相同，Java 中的流程控制结构主要有顺序结构、选择结构及循环结构三种。

顺序结构即是按照从上到下的顺序执行语句，没有转移及重复。

选择结构是根据给定的条件成立与否，执行不同的语句或语句组。Java 的选择结构主要有二路选择结构(if 选择结构)及多路选择结构(switch 选择结构)两种。

循环控制结构是在一定的条件下，反复执行某段程序的流程结构，被反复执行的程序称为循环体。循环控制结构是程序中非常重要和最基本的一种结构，它是由循环语句来实现的。一个循环一般包括 4 部分的内容。

(1) 初始化部分：用来设置循环的一些初始条件，如计数器清零等。

(2) 循环体部分：这是反复执行的一段代码，可以是单一的一条语句，也可以是复合语句。

(3) 迭代部分：这是当前循环结束，下次循环开始执行的语句，常常用来使计数器进行增减操作。

(4) 终止部分：通常是布尔表达式，每一次循环要对该表达式求值，以验证是否满足循环终止条件。

Java 中提供的循环语句共有三种：

(1) for 语句。

(2) while 语句。

(3) do-while 语句。

除此以外，还有与程序转移有关的跳转语句，主要用于循环中，用于实现循环执行过程中的流程转移。为了提高程序的可靠性和可读性，Java 语言不支持无条件跳转的 goto 语句。循环中常用的跳转语句有两个：break 和 continue 语句。

下面分别对控制结构及跳转语句进行介绍。

3.1.1 if 选择结构

if 语句是 Java 程序中最常见的分支控制语句。if 语句的语法形式如下：

(1) 只有 if 语句，即只有 if 分支，没有 else 分支。如果条件表达式成立，则执行 if 分支，其语法形式如下：

```
if (boolean-expression)  {
   //statement1;
}
```

(2) if-else 语句根据判定条件的真假来执行两种操作中的一种，其一般的格式为：

```
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，根据这个表达式的值是“真”或“假”来决定选择哪个分支来执行。执行 if 语句时，程序先计算条件表达式的值，如果为“真”，则执行 if 分支语句组；如果为“假”，则执行 else 分支语句。

注意：这里的分支语句组如果只有一个语句，如一个单独的赋值语句或另一个完整的 if-else 语句，则不需要用大括号括起；否则分支中的所有语句都需要用大括号括起，构成复合语句。

(3) if-else 语句的一种特殊形式为：

```
if (boolean-expression1) {
//statement1;
     }
else if (boolean-expression2) {
    //statement2;
```

```
    }
          …
else {
//statement n;
   }
```

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

例 3-1　计算每个月的天数。

假设暂不考虑闰年，则二月份为 28 天。已知月份，求其天数的方法如下：

```
static int daysInMonth(int month) {
    if (month==2)
       return(28);
    else  if ((month==4)||(month==6)||(month==9)||(month==11))
          return(30);
    else  return(31);
}
```

在上面的程序中，当变量 month 的值为 2 时，返回 28；如果变量 month 的值为 4、6、9、11 时，返回 30；否则，返回 31。

3.1.2　switch 选择结构

switch 语句是多分支的选择结构，它的一般格式如下：

```
switch (switch-expression) {
    case value1:   statements for case1; break;
    case value2:   statements for case2; break;
    …
    case valueN:   statements for caseN; break;
    default:       statements for default case; break;
}
```

switch 语句中表达式的值(switch－expression)必须是整形或字符型；常量值 value1 到常量值 valueN 也必须是整型或字符型。Switch 语句首先计算表达式的值，如果表达式的值和某个 case 后面的值相同，则从该 case 之后开始执行，直到 break 语句为止。若没有一个常量与表达式的值相同，则从 default 之后开始执行。Default 是可有可无的，如果它不存在，并且所有的常量值都和表达式不相同，那么 switch 语句就不会进行任何处理。另外，在同一个 switch 语句中，case 后的值必须互不相同，但是次序没有要求。

例 3-2　计算每个月的天数。

下面使用 switch 结构计算每个月的天数，方法声明如下：

```
static int daysInMonth(int month) {
```

```
    int days;
    switch(month) {
        case 2: days=28; break;
        case 4:
        case 6:
        case 9:
        case 11: days=30; break;
        default: days=31;
    }
    return(days);
}
```

在上面的例子中,switch 语句首先判断月份,然后根据月份的值来计算天数,假设变量 month 的值是 2,则执行完 switch 语句后,变量 days 的值为 28。

需要注意的是,switch 语句的每一个 case 判断,都只负责指明分支的入口点,而不指定分支的出口点。分支的出口点需要编程人员用相应的跳转语句 break 来标明。

如果将上面程序中的 break 语句去掉,考虑下面例子的执行结果。

```
static int daysInMonth(int month) {
    int    days;
    switch(month) {
        case 2: days=28;
        case 4:
        case 6:
        case 9:
        case 11: days=30;
        default: days=31;
    }
    return(days);
}
```

假设变量 month 的值还为 2,执行完 switch 语句后,变量 days 的值被置成什么呢?是 31,而不是 28? 为什么? 因为 case 判断只负责指明分支的入口点,表达式的值与第一个 case 分支的判断值相匹配后,程序的流程进入第一个分支,将 days 的值置为 28。由于没有专门的分支出口,所以流程将继续沿着下面的分支逐个执行,days 的值被置为 30,最后置为 31。所以如果希望程序的逻辑结构正常完成分支的选择,需要为每一个分支另外编写退出语句。如例 3-2 所示那样,通过 break 语句来正常退出 case 分支。

也可以使用 return 语句返回结果,并结束方法的调用。例 3-2 中的方法可简化成下面的代码,也能得到正确的结果。

```
static int daysInMonth(int month) {
    switch(month) {
        case 2: return(28);
        case 4:
```

```
        case 6:
        case 9:
        case 11: return(30);
        default: return(31);
    }
}
```

例 3-3　已知一个学生的分数(带有小数点),给出其分数等级。

90—100 分为 A 级;80—89 分为 B 级;70—79 分为 C 级;60—69 分为 D 级;0—59 分为 E 级。

下面编写一个类方法进行转换。由于 switch 后的表达式需要为整数类型或字符型,因而需要对分数进行运算,使得相同等级的分数尽量对应到相同的数字,而不同等级的分数对应到不同的数字。显然,在这个问题中,应将分数除以 10 再去掉小数部分,就得到了整数 10 到 0。这个功能可调用 Math 类中的静态方法 floor()方法来实现。floor 方法返回不大于其参数的最大整数,但其返回类型仍为实型。

为了直接使用 floor 方法,在系统中静态引入 java.lang.Math 包下的所有静态成员。由于一共只有五个等级,并且这五个等级是确定的,分别为 A、B、C、D、E,因此可以使用枚举类型来实现等级。程序如下:

```
import static java.lang.Math.*;     //静态引入 java.lang.Math 包下的静态成员
enum Grade {                        //枚举类型,表示等级,一共有五个等级
    A,
    B,
    C,
    D,
    E;
}
public class Score {
    public static Grade gradeLevel(double g){
    int n=(int)floor(g/10);
    switch (n) {
        case 10:
        case 9:return Grade.A;
        case 8:return Grade.B;
        case 7:return Grade.C;
        case 6:return Grade.D;
        default: return Grade.E;
        }
    }
    public static void main(String[] args){
        System.out.println("gradeLevel(100)="+gradeLevel(100));
        System.out.println("gradeLevel(95.5)="+gradeLevel(95.5));
        System.out.println("gradeLevel(88)="+gradeLevel(88));
```

```
        System.out.println("gradeLevel(72)="+gradeLevel(72));
        System.out.println("gradeLevel(68.5)="+gradeLevel(68.5));
        System.out.println("gradeLevel(60)="+gradeLevel(60));
        System.out.println("gradeLevel(59.5)="+gradeLevel(59.5));
        System.out.println("gradeLevel(35)="+gradeLevel(35));
    }
}
```

运行结果如下:

```
gradeLevel(100)=A
gradeLevel(95.5)=A
gradeLevel(88)=B
gradeLevel(72)=C
gradeLevel(68.5)=D
gradeLevel(60)=D
gradeLevel(59.5)=E
gradeLevel(35)=E
```

3.1.3 for 循环结构

for 循环结构是 Java 语言三个循环语句中功能较强、使用较广泛的一个,它的一般语法格式如下:

```
for (start-expression; check-expression, update-expression) {
    //body of the loop;
}
```

start-expression 完成循环变量和其他变量的初始化工作;check－expression 是返回布尔值的条件表达式,用于判断循环是否继续;update－expression 用来修整循环变量,改变循环条件。

注意:三个表达式之间用分号隔开。

语句 for 的执行过程是这样的:首先根据初始表达式 start-expression 完成必要的初始化工作;再判断表达式 check-expression 的值,若为真,则执行循环体,执行完循环体后再返回表达式 update-expression,计算并修改循环条件,这样一轮循环就结束了。第二轮循环从计算并判断表达式 check-expression 开始,若表达式的值仍为真,则循环继续,否则跳出整个 for 语句执行 for 循环下面的句子。

下面语句是用 for 循环语句实现打印五行“**********”。

```
for (int i=1; i<=5; i++)
    System.out.println("**********");
```

在上面的例子中,完成初始化工作的表达式定义了一个整型局部变量 i,这种情况下,变量 i 的生命周期仅限于 for 语句及其循环体中,一旦程序的执行跳出了 for 循环,变量 i

就不能再使用了。

另外，for 语句的三个表达式都可以为空，但是其间的分号不能省略。若表达式 check-expression 为空，则表示当前循环是一个无限循环，需要在循环体中增加跳转语句终止循环。

For 循环的循环体中也可以包含另一个或多个 for 循环，这称为 for 循环的嵌套。

例 3-4　打印九九乘法表。

程序如下：

```
public class MultiTable {
    public static void main(String[] args){
        for (int i=1; i<=9;i++) {
            for (int j=1; j<=i;j++)
                System.out.print("  "+i+"*"+j+"="+i*j);
            System.out.println();
        }
    }
}
```

在 Java 5 中提供了增强 for 循环的功能。增强 for 循环可以用来对数组或者集合对象进行遍历，其语法格式如下：

```
for (Type name:Array/Set) {
    //body of the loop;
}
```

例 3-5　使用增强 for 循环打印星期一到星期日的英文名。

```
public class PrintDay {
    public static void main(String[] args){
        String days[]={"Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
        "Saturday", "Sunday"};
        for (String day:days)
            System.out.print(day+" ");
        System.out.println();
    }
}
```

运行结果如下：

```
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

3.1.4　while 语句

while 语句实现“当型”循环，其一般语法格式如下：

```
while (check-expression) {
    //body of the loop;
}
```

其中条件表达式(check-expression)的返回值为布尔型,循环体可以是单个语句,也可以是复合语句块。

while 语句的执行过程是先判断条件表达式(check-expression)的值,若为真,则执行循环体,循环体执行完后再无条件转向条件表达式做计算与判断;当计算出条件表达式的值为假时,跳过循环体执行 while 语句后面的语句。为真,则继续执行循环。如:

```
char ch='a';
while (ch!='\n'){
    System.out.println(ch);
    ch=(char)System.in.read();              //接受键盘输入
}
```

此例是循环接受并输出从键盘输入的字符,直到输入的字符为回车为止。

例 3-6 计算存款收益。

假设银行中存款 10 000 元,按 11.25%的利率,一年后连本带利将变为 11 125 元。你若将此款继续存入银行,试问多长时间就会连本带利翻一番?

假设上一年度的存款额为 x,则下一年度的存款额为 $x+x\times 11.25\geqslant x$。如果 x 的值没有翻一番,则继续使用此公式计算下一年度的存款额。使用 while 循环,程序如下:

```
import java.text.*;
public class CalProfit {
    public static void main(String[] args){
        double original,money,interest;
        int years=0;
        original=money=10000;
        interest=11.25/100;
        System.out.println("year\tmoney");
        while (money<2*original) {
            years=years+1;
            money=money+(interest*money);
            System.out.printf("%d\t%.2f\n", years, money);
        }
        System.out.println();
        System.out.printf("第%d年存款额达到%.2f元\n", years, money);
    }
}
```

运行结果如下：

```
year        money
1        11125.00
2        12376.56
3        13768.93
4        15317.93
5        17041.20
6        18958.33
7        21091.14

第 7 年存款额达到 21091.14 元
```

3.1.5　do-while 语句

do-while 语句实现“直到型”循环，它的一般语法结构如下：

```
do {
    //body of the loop;
} while (check-expression);
```

do-while 语句的使用与 while 语句很类似，不同的是它不像 while 语句是先计算条件表达式的值，而是无条件地执行一遍循环体，再来判断条件表达式的值。若表达式的值为真，则再运行循环体；否则跳出 do-while 循环，执行下面的语句。可以看出，do-while 语句的特点是它的循环体将至少要执行一次。

3.1.6　break 语句

break 语句可用于三种情况：

(1) 在 switch 结构中，break 语句用来终止 switch 语句的执行。

(2) 在 for 循环及 while 循环结构中，用于终止 break 语句所在的最内层循环。

例 3-7　简单 break 应用举例。

程序如下：

```
public class BreakTest {
    public static void main(String args[]) {
        String output="";
        int i;
        for (i=1; i <=10; i++) {
            if (i ==5) {
                break; //break loop only if count ==5
            }
            output+=i+" ";
        }
```

```
        output+="\nBroke out of loop at i="+i;
        System.out.println(output);
    }
}
```

运行结果如下：

```
1 2 3 4
Broke out of loop at i=5
```

从运行结果看到：当执行 break 语句时，程序流程跳出 for 循环。

Break 语句也可以用在嵌套的循环结构中，此时执行 break 语句将跳出 break 所在的最内层循环。

例 3-8 在嵌套循环中使用 break 语句。

例 3-4 的九九乘法表也可以使用下面的程序来实现。

```
public class MultiTable {
    public static void main(String[] args) {
        for (int i=1; i <=9; i++) {
            for (int j=1; j <=9; j++) {
                if (j >i) {
                    break;
                }
                System.out.printf("  %d * %d=%d", i, j, i * j);
            }
            System.out.println();
        }
    }
}
```

运行结果如下：

```
1 * 1=1
2 * 1=2  2 * 2=4
3 * 1=3  3 * 2=6   3 * 3=9
4 * 1=4  4 * 2=8   4 * 3=12  4 * 4=16
5 * 1=5  5 * 2=10  5 * 3=15  5 * 4=20  5 * 5=25
6 * 1=6  6 * 2=12  6 * 3=18  6 * 4=24  6 * 5=30  6 * 6=36
7 * 1=7  7 * 2=14  7 * 3=21  7 * 4=28  7 * 5=35  7 * 6=42  7 * 7=49
8 * 1=8  8 * 2=16  8 * 3=24  8 * 4=32  8 * 5=40  8 * 6=48  8 * 7=56  8 * 8=64
9 * 1=9  9 * 2=18  9 * 3=27  9 * 4=36  9 * 5=45  9 * 6=54  9 * 7=63  9 * 8=72  9 * 9=81
```

break 语句还可以与标号一同使用，当与标号一同使用时，执行 break 语句将跳出标号所标识的循环。