第3章 Java语言面向对象特性

本章将介绍三方面内容：①Java语言中类和对象的定义；②Java语言对OOP(Object Oriented Programming)的3个主要特性——封装、继承和多态的支持机制；③数组对象这种数据结构。面向对象是Java语言的基本特性之一，深刻理解这个特性是学好Java语言程序设计的关键。

3.1 类与对象

类描述了同一类对象共同拥有的数据和行为，包含被创建对象的属性和方法的定义。学习Java语言编程就是学习怎样编写类，也就是怎样利用Java语言的语法描述一类事物的公共属性和行为。在Java语言中，对象的属性通过变量描述，对象的行为通过方法实现。方法可以操作属性以形成一个算法实现一个具体的功能，把属性和方法封装成一个整体就形成了一个类。

3.1.1 类与对象的定义

Java程序是由一个或若干个类组成的，类是Java程序的基本组成单位。编写Java程序就是定义类，然后再根据定义的类创建对象。类由成员变量和成员方法两部分组成，成员变量的类型可以是基本数据类型、数组类型、自定义类型，成员方法用于处理类的数据。一个Java类从结构上可以分为类的声明和类体两部分，如图3.1所示。

1. 类的声明

类的声明用于描述类的名称以及类的属性(如访问权限、与其他类的关系等)。声明类的语法如下：

```
[public] [abstract | final] class <className> [extends
superClassName] [implements interfaceNameList] {...}
```

- “[ ]”：表示可选项，“< >”表示必选项，“|”表示多选一。

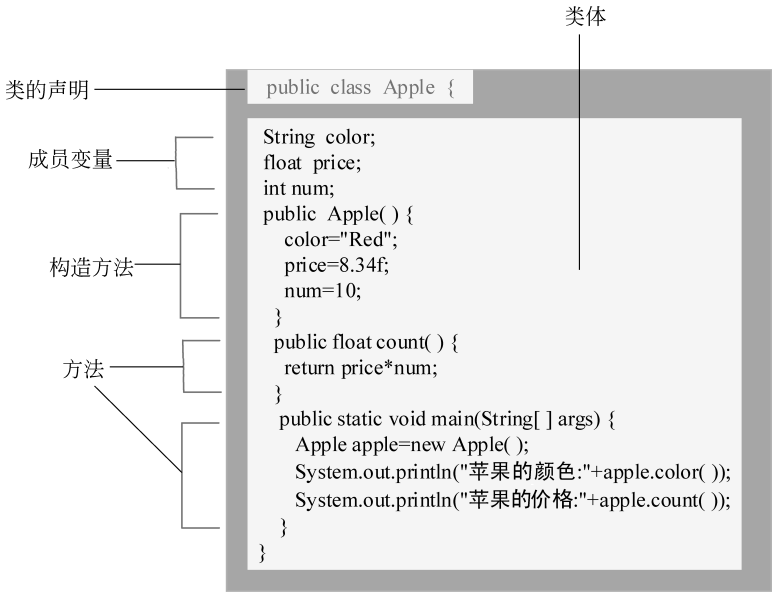

图 3.1 类定义的结构示意图

- public、abstract 或 final：指定类的访问权限及其属性，用于说明所定义类的相关特性(后续章节介绍)。
- class：Java 语言的关键字，表明这是一个类的定义。
- className：指定类名称的标识符。
- extends super ClassName：指定所定义的类继承自哪个父类。若使用 extends 关键字，则父类名称为必选参数。
- Implements interfaceNameList：指定该类实现哪些接口。当使用 implement 关键字时，接口列表为必选参数。

2. 类体

类体指的是出现在类声明后面的花括号中的内容。类体提供了类的对象在生命周期中需要的所有代码——①构造和初始化新对象的构造方法；②表示类及其对象状态的变量；③实现类及其对象的方法；④进行对象清除的 finalize()方法。

3.1.2 成员变量与局部变量

当一个变量的声明出现在类体中，并且不属于任何一个方法时，该变量称为类的成员变量。在方法体中声明的变量以及方法的参数统称为方法的局部变量。

1. 成员变量

成员变量表示类的状态和属性，其声明的语法如下：

```
[public | protected | private ] [static] [final] [transient] [volative]<type>
<variableName>;
```

- public、protected 或 private：指定变量的访问权限。

- static：指定变量为静态变量(也称为类变量)，其特点是可以通过类名直接访问。如果省略，则表示为实例变量。
- final：指定变量为常量。
- transient：声明变量为暂时性变量，告知 JVM 该变量不属于对象的持久状态，从而不能被持久存储。如果省略，则类中的所有变量都是对象持久状态的一部分，当对象被保存到外存时，这些变量必须同时被保存。
- volatire：指定变量在被多个并发线程共享时，JVM 将采取优化的控制方法提高线程的并发执行效率。该修饰符是 Java 语言的一种高级编程技术，一般程序员很少使用。
- type：指定变量的数据类型。
- variableName：指定变量的名称。

【例 3.1】　在类 Apple 的定义中，声明 3 个类的成员变量，并在 main()方法中通过输出它们的值说明其状态特征。

```
1  public class Apple {
2    public String color;                    //公共变量 color
3    public static int num;                  //静态变量 num
4    public final boolean MATURE=true;       //定义常量 MATURE 并赋值
5    public static void main(String[ ] args) {
6      System.out.println("苹果的数量: "+Apple.num);
7      Apple apple=new Apple();
8      System.out.println("苹果的颜色: "+apple.color);
9      System.out.println("苹果是否成熟: "+apple.MATURE);
10    }
11 }
```

【运行结果】

```
E:\Java\JNBExamples>java Apple
苹果的数量: 0
苹果的颜色: null
苹果是否成熟: true
```

【分析讨论】

- num 是静态变量(类变量)，在运行时 JVM 只为类变量分配一次内存，并在加载类过程中完成其内存分配，所以可以通过类名直接访问(第 6 行语句)。
- color 与 MATURE 都是实例变量，必须通过创建对象的名称 apple 访问(第 7、8、9 行语句)。

2. 局部变量

局部变量作为方法或语句块的成员，存在于方法的参数列表和方法体的定义中。其定义的语法如下：

```
[final]  <type> <变量名字>
```

- final：可选项，指定局部变量为常量。
- type：指定局部变量的数据类型，它可以是任意一种 Java 语言的数据类型。
- 变量名：变量名必须是合法的 Java 语言标识符。
- 对于类中定义的成员变量，如果没有初始化，那么 Java 语言将自动给它们赋予一个初值，即默认初始值。而对于局部变量，在使用之前必须进行初始化，然后才能使用。

【例 3.2】 在类 Apple 的定义中，解释使用局部变量要注意的问题。

```
1   public class Apple {
2     String color="Red";      //成员变量 color,赋初值"Red"
3     float price;             //成员变量 price,默认初始值 0.0f
4     public String getColor() {
5       return color;       }
6     public float count() {
7       int num;               //局部变量 num
8       if(num<0)              //错误语句,因为局部变量 num 还没有被赋值就使用
9         return 0;
10      else
11        return price * num;
12    }
13    public static void main(String[] args) {
14      Apple apple=new Apple();
15      System.out.println("苹果总价格: "+apple.count());
16    }
17  }
```

【编译结果】

```
E:\Java\JNBExamples>javac Apple.java
Apple.java:8: 错误: 可能尚未初始化变量 num
    if(num<0)                  //错误语句,因为局部变量 num 还没有被赋值就使用
       ^
```

1 个错误。

【分析讨论】

- 在程序的第 15 行，通过对象 apple 调用了方法 count()，而此时在 count()方法中定义的局部变量 num 在使用之前没有进行初始化，所以造成程序编译错误(第 8 行语句)。

3. 变量的有效范围

变量的有效范围是指变量在程序中的作用区域，在区域外不能直接访问变量。有效范围决定了变量的生存周期——指从声明一个变量并分配内存空间、使用变量开始，然后释放变量并清除所占用内存空间的过程。变量声明的位置，决定了变量的有效范围。根据变量的有效范围的不同，可以将变量分为成员变量和局部变量两种。

- 成员变量：类体中声明的成员变量在整个类的范围内有效。
- 局部变量：在方法内或方法内的语句块(方法内部，“{”与“}”之间的代码块)中声明的变量称为局部变量。在语句块以外，方法体内声明的变量在整个方法内有效。

【例 3.3】　在类 Olympics1 的定义中，说明成员变量与局部变量的有效范围。

```
1   public class Olympics1 {
2    private int medal_All=800;          //成员变量
3     public void China() {
4       int medal_CN=100;                //代码块外、方法体内的局部变量
5       if(medal_CN<1000) {              //代码块
6         int gold=50;                   //代码块的局部变量
7         medal_CN+=30;                  //允许访问本方法的局部变量
8         medal_All-=130;                //允许访问本类的成员变量
9       }                                //代码块结束
10    }
11  }
```

【分析讨论】

- 在第 5 行语句中，允许访问类的成员变量 medal_All 和在方法中定义的局部变量 medal_CN。

3.1.3　成员方法

类的成员方法由方法声明和方法体两部分组成，其语法如下：

```
[accessLevel][static][final][abstract][synchronized]<return_type><name>
([<argument_list>])[throws<exception_list>]{  [block]  }
```

- accessLevel：方法的访问权限，可选值为 public、protected 与 private。
- static：指定成员方法为静态方法。
- final：指定成员方法为最终方法。
- abstract：指定方法为抽象方法。
- synchronized：控制多个并发线程对共享数据的访问。
- return_type：确定方法的返回类型，可以是任意的 Java 语言数据类型。如果方法没有返回值，则可以指定为 void 标识。
- name：成员方法的名称。
- argument_list：形式参数列表。方法可分为有参数和无参数两种。参数类型可以是 Java 语言的数据类型。
- throws ＜exception_list＞：列出方法将要抛出的异常。
- block：方法体包括局部变量的声明和所有合法的 Java 语句。方法体可以省略，但是外面的一对花括号不能省略。
- 方法体中的局部变量作用域只在方法内部，当方法调用返回时，局部变量也不再存在。

- 如果局部变量的名字和所在类的成员变量的名字相同，则类的成员变量被隐藏；如果要将成员变量显式地表现出来，则需要在成员变量的前面加上关键字 this。

【例 3.4】 在类 Olympics2 的定义中，说明在成员变量与局部变量同名的情形下，用 this 标识成员变量的方法。

```
1   public class Olympics2 {
2     private int gold=0;
3     private int silver=0;
4     private int copper=0;
5     public void changeModel(int a,int b,int c) {
6       gold=a;
7       int silver=b;          //silver 使同名类成员变量隐藏
8       int copper=50;         //copper 使同名类成员变量隐藏
9       System.out.println("In changeModel: "+"金牌="+gold+" 银牌="+silver+"
        铜牌="+copper);
10      this.copper=c;         //给类成员变量 copper 赋值
11    }
12    String getModel() {
13      return "金牌="+gold+" 银牌="+silver+" 铜牌="+copper;
14    }
15    public static void main(String args[]) {
16      Olympics2 o2=new Olympics2();
17      System.out.println("Before changeModel: "+o2.getModel());
18      o2.changeModel(100,100,100);
19      System.out.println("After changeModel: "+o2.getModel());
20    }
21  }
```

【运行结果】

```
E:\Java\JNBExamples>java Olympics2
Before changeModel: 金牌=0 银牌=0 铜牌=0
In changeModel: 金牌=100 银牌=100 铜牌=50
After changeModel: 金牌=100 银牌=0 铜牌=100
```

【分析讨论】

- 在 main()方法中，创建了类 Olympics2 的对象 o2。第 17 行语句通过 o2 调用了 getModel()方法。getModel()方法中操作的全部是成员变量，而且第 2～4 行语句的成员变量进行的是显式初始化，所以得到第一行的输出结果。
- 成员变量 silver 和 copper 与方法 changeModel()中定义的局部变量同名，如果不加特殊标识 this，则在方法 changeModel()中操作的是局部变量 silver 和 copper。因此，在第 18 行语句调用 changeModel()方法时，得到第三行的输出结果。
- changeModel()方法更新了成员变量 gold 和 copper 的值，所以第 19 行语句在调用 getModel()方法时，能得到第四行的输出结果。

注意：return 通常放在方法的最后，用于退出当前方法并返回一个值，使程序把控制权交给调用它的语句。return 语句中的返回值必须与方法声明中的返回值类型相匹配。

3.1.4　对象的创建

在 Java 语言中，对象是通过类创建的，是类的动态实例。一个对象在程序运行期间的生存周期包括创建、使用和销毁 3 个阶段。Java 语言的对象创建、使用和销毁有一套完善的机制。在 Java 语言中，创建一个对象的语法如下：

```
<className> <objectName>
```

- className：指定一个已经定义的类。
- objectName：指定一个对象的名称。

例如，声明 Apple 类的一个对象 redApple 的语句如下：

```
Apple redApple;
```

声明对象时，只是在内存中为其分配一个引用空间，并设置为 null，表示不指向任何存储空间，然后为对象分配存储空间。这个过程称为对象的实例化。实例化对象使用关键字 new 实现，它的语法如下：

```
<objectName>=new <SomeClass>([argument_list]);
```

- objectName：指定已经声明的对象名称。
- SomeClass：指定需要调用的构造方法名称。
- srgument_list：指定构造方法的入口参数。如果无参数，则可省略。

在声明 Apple 类的一个对象 greenApple 后，通过下面的语句可为对象 greenApple 分配存储空间，执行 new 运算符后的构造方法将完成对象的初始化，并返回对象的引用。当对象创建不成功时，new 运算符将返回 null 给变量 redApple。

```
greenApple=new Apple();
```

声明对象时，也可以直接实例化对象，即把上述步骤合二为一。

```
Apple greenApple=new Apple();
```

【例 3.5】 在类 Point 中定义两个成员变量，在构造方法中定义具有两个整数的参数列表。

```
1  public class Point {
2   int x=1;
3   int y=1;
4   public Point(int x,int y) {
5     this.x=x;
6     this.y=y;
7   }
8  }
```

如果执行如下语句，则可以创建 Point 类的对象。

```
Point pt=new Point(2,3);
```

下面是上述语句的对象创建与初始化过程。

(1) 声明一个 Point 类的对象 pt，并为其分配一个引用空间，初始值为 null。此时，引用没有指向任何存储空间，即没有分配存储地址。

(2) 为对象分配存储空间，并将成员变量进行默认初始化，数值型变量的初值为 0，逻辑型变量的初值为 false，引用型变量的初值为 null。

(3) 执行显式初始化，即执行在类成员变量声明时带有的简单赋值语句。

(4) 执行构造方法，进行对象的初始化。

(5) 最后，执行语句中的赋值操作，将新创建对象的存储空间的首地址赋给 pt 的引用空间。

图 3.2 所示为执行上述过程中 5 个步骤时的对象状态。

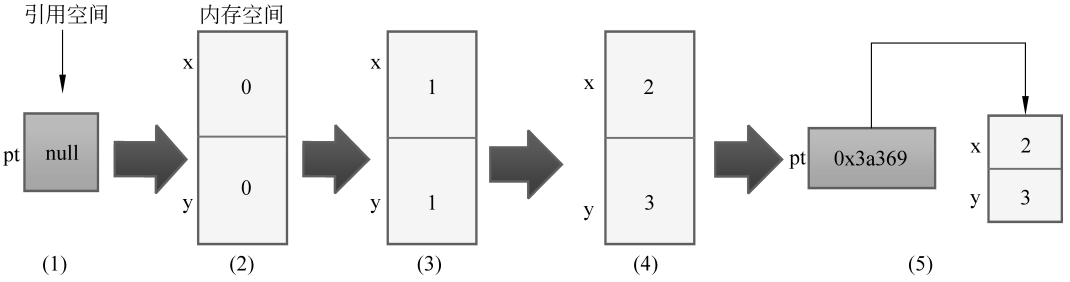

图 3.2 对象创建与初始化的示意图

3.1.5 对象的使用

创建对象以后，可以通过“.”操作符对成员变量进行访问。访问对象的成员变量的语法如下：

```
objectReference.variableName;
```

objectReference：指定调用成员变量的对象名称。

variableName：指定要调用的成员变量的名称。

一般地，不提倡通过对象对成员变量进行直接访问。规范的对象变量访问方式是通过对象提供的统一接口 setter 和 getter(即成员方法)对变量进行读写操作，其优点是可以实现变量的正确性、完整性的约束检查。当需要对对象变量进行直接访问时，可以使用 Java 语言的访问控制机制，以控制哪些类能够直接对变量进行访问。

调用对象的成员方法的语法如下：

```
objectReference.methodName([argument_list]);
```

objectReference：指定调用成员方法的对象名称。

methodName：指定要调用的成员方法的名称。

argument_list：指定被调用的成员方法的参数列表。

对象的方法可以通过设置访问权限允许或禁止其他对象访问。

【例 3.6】　创建 Point 类的对象 pt，访问其成员方法和成员变量。

```
1   public class Point {
2     int x=1;
3     int y=1;
4     public void setXY(int x,int y) {
5         this.x=x;
6         this.y=y;
7     }
8     public int getXY() {
9         return x * y;
10    }
11    public static void main(String[ ] args) {
12       Point pt=new Point();          //声明并创建 Point 类的对象 pt
13       pt.x=2;                        //访问对象 pt 的成员变量 x,并改变其值
14       System.out.println("x 与 y 的乘积为: "+pt.getXY());
15       pt.setXY(3,2);                 //调用对象 pt 带参数的成员方法 setXY()
16       System.out.println("x 与 y 的乘积为: "+pt.getXY());  //调用成员方法 getXY()
17    }
18  }
```

【运行结果】

```
E:\Java\JNBExamples>java Point
x 与 y 的乘积为: 2
x 与 y 的乘积为: 6
```

【分析讨论】

- 在 main()方法中，创建了类 Point 的对象 pt(行 12 行语句)，通过 pt 修改了 x 的值(第 13 行语句)。
- 第 14 行语句通过调用方法 getXX()输出更新前的执行结果。第 15 行语句通过访问 setXY()方法，传递参数 3 和 2 到变量 x 和 y，即将成员变量更新。最后，再次调用 getXY()输出更新后的执行结果(第 16 行语句)。

3.1.6　对象的销毁

在 Java 语言中，程序员可以创建所需要的对象，但是不必关心对象的删除，因为 Java 语言提供了垃圾回收机制，可以自动地判断对象是否还在使用，并能够自动销毁不再使用的对象，回收对象所占的系统资源。Object 类提供了 finalize()方法，自定义的 Java 类可以覆盖这个方法，并在这个方法中释放对象所占的资源。JVM(Java 虚拟机)的垃圾回收操作的生命周期如图 3.3 所示。

在 JVM 垃圾回收器看来，存储空间中的每个对象都可能处于以下 3 种状态之一。

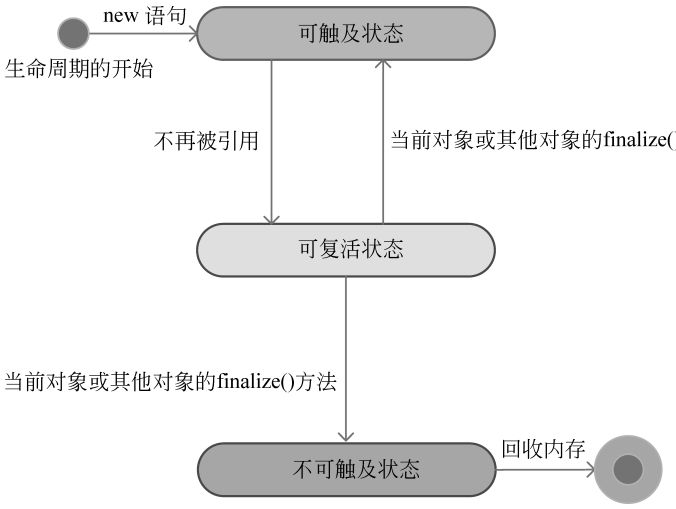

图 3.3 JVM 的垃圾回收操作的生命周期

- 可触及状态：当一个对象被创建之后，只要程序中还有引用变量在引用它，那么它就始终处于可触及状态。
- 可复活状态：当程序不再有任何引用变量对象时，就进入可复活状态。在这个状态中，垃圾回收器将会释放它占用的存储空间。在释放之前，将会调用它及其处于可复活状态的对象的 finalize()方法。这些 finalize()方法有可能使对象重新转到可触及状态。
- 不可触及状态：JVM 执行完所有可复活对象的 finalize()方法之后，假如这些方法都没有使该对象转到可触及状态，那么该对象将进入不可触及状态。只有当对象处于不可触及状态时，垃圾回收器才真正回收它占用的存储空间。

3.1.7 方法重载

当在同一个类中定义了多个同名而内容不同的成员方法时，则称这些方法为重载方法(overloading method)。重载方法通过形式参数列表中参数个数、参数类型和参数顺序的不同加以区分。在编译阶段，Java 语言的编译器要检查每个方法所用的参数数量和类型，然后调用正确的方法，即实现 Java 语言的编译时多态。Java 语言规定重载方法必须遵循以下原则。

- 方法的参数列表必须不同，包括参数的个数或类型，以此区分不同的方法体。
- 方法的返回值类型、修饰符可以相同，也可以不同。
- 在实现方法重载时，方法返回值的类型不能作为区分方法重载的标志。

【例 3.7】 在类 Calculate 的定义中，定义两个名称为 getArea()的方法(参数个数不同)和两个名称为 draw()的方法(参数类型不同)，用以输出不同图形的面积。

```
1   public class Calculate {
2     final float PI=3.14159f;
```

```
3     public float getArea(float r) {            //计算面积的方法
4       return PI * r * r;
5     }
6     public float getArea(float l,float w) {  //重载方法 getArea()
7       return l * w;
8     }
9     public void draw(int num) {                //画任意形状的图形
10      System.out.println("画"+num+"个任意形状的图形");
11    }
12    public void draw(String shape) {           //画指定形状的图形
13      System.out.println("画一个"+shape);
14    }
15    public static void main(String[] args) {
16      Calculate c=new Calculate();             //创建 Calculate 类的对象
17        float l=20;
18        float w=40;
19        System.out.println("长为"+l+"宽为"+w+"的矩形面积是: "+c.getArea(l,w));
20        float r=6;
21        System.out.println("半径为"+r+"的圆形面积是: "+c.getArea(r));
22        int num=8;
23        c.draw(num);
24        c.draw("矩形");
25    }
26  }
```

【运行结果】

```
E:\Java\JNBExamples>java Calculate
长为 20.0 宽为 40.0 的矩形面积是: 800.0
半径为 6.0 的圆形面积是: 113.097244
画 8 个任意形状的图形
画一个矩形
```

【分析讨论】

- 在第 19 行语句中调用了 getArea()方法，由于传递的实际参数是两个 float 类型变量，所以此时对象 c 调用的是第 6～8 行语句定义的 getArea()方法。
- 在第 21 行语句中调用的 getArea()方法，由于传递的实际参数是一个 float 类型的变量 r，所以此时对象 c 调用的是第 3～5 行语句定义的 getArea()方法。
- 在第 23 行语句中调用的 draw()方法，由于传递的实际参数是一个 int 型变量 num，所以此时对象 c 调用的是第 9～11 行语句定义的 draw()方法。
- 第 24 行语句中调用的 draw()方法，由于传递的是一个 String 类型的参数，所以此时对象 c 调用的是第 12～24 行语句中定义的 draw()方法。

3.1.8　关键字 this

关键字 this 表示对象本身，常用于一些容易混淆的情形。例如，当成员方法的形式

参数名称与数据所在类的成员变量名称相同时，或者当成员方法的局部变量名称与类的成员变量名称相同时，在方法内部可以借助关键字 this 指明引用的是类的成员变量，而不是形式参数或局部变量，从而提高程序的可读性。

this 代表了当前对象的一个引用，可以将其理解为对象的另外一个名字，通过这个名字可以顺利地访问对象，修改对象的数据成员，调用对象的方法。归纳起来，this 的使用情形有如下 3 种。

- 用来访问当前对象的一个引用，使用格式为：this.数据成员。
- 用来访问当前对象的成员方法，使用格式为：this.成员方法(参数列表)。
- 重载构造方法时，用来引用同类的其他构造方法，使用格式为：this.(参数列表)。

【例 3.8】 通过关键字 this 区别成员变量 color 和局部变量 color，并通过 this 访问当前对象的成员方法 count()。

```
1   public class Fruit {
2     String color="绿色";
3     double price;
4     int num;
5     public void harvest() {
6       String color="红色";
7       //此时输出的是成员变量 color
8       System.out.println("水果原来是: "+this.color+"的!");
9       System.out.println("水果已经收获!");
10      System.out.println("水果现在是: "+color+"的!");   //此时输出的是局部变量
11      //使用 this 调用成员方法 count()
12      System.out.println("水果的总价格是: "+this.count(2.14,50)+"元。");
13    }
14    public double count(double price, int num) {
15      this.price=price;              //将形参 price 赋值给成员变量 price
16      this.num=num;                  //将形参 num 赋值给成员变量 num
17      return price * num;
18     }
19     public static void main(String[] args) {
20      Fruit obj=new Fruit();
21      obj.harvest();
22     }
23  }
```

【运行结果】

```
E:\Java\JNBExamples>java Fruit
水果原来是: 绿色的!
水果已经收获!
水果现在是: 红色的!
水果的总价格是: 107.0元
```

【分析讨论】

- 在方法 count(double price,int num)中,如果不使用 this,则作为类的成员变量的 price 和 num 将被隐藏,将不会得到预期的对象初始化结果。
- 在方法 harvest()中,使用 this 调用方法 count()的语句(第 12 行语句),这个 this 的使用是不必要的。当一个对象的方法被调用时,Java 语言会自动给对象的变量和方法都加上 this 引用,指向内存中的对象,所以有些情形下不需要使用 this 关键字。

3.1.9　构造方法

Java 语言中所有的类都有构造方法,用于对象的初始化。构造方法也有名称、参数、方法体以及访问权限的限制。

1. 构造方法的声明

定义构造方法的语法如下所示:

```
[accessLevel]<className>([argument_list]) {
    [block]
}
```

- accessLevel:指定构造方法的访问权限。
- className:指定构造方法的名称,这个名称必须与所属类的名称相同。
- argument_list:指定构造方法中所需要的参数。
- block:方法体是构造方法的实现部分,包括局部变量的声明和所有合法的 Java 语句。当方法体省略时,其外面的一对花括号不能省略。

构造方法与一般方法在声明上的区别如下。

- 构造方法的名字必须与类名相同,并且构造方法不能有返回值。
- 用户不能直接调用构造方法,必须通过关键字 new 自动调用。

【例 3.9】　在类 Apple 的定义中,声明两个构造方法,并通过这两个构造方法分别创建两个 Apple 类的对象。

```
1   public class Apple {
2     private int num;
3     private double price;
4     public Apple() {
5       num=10;
6       price=2.34; }
7     public Apple(int num,double price) {
8       this.num=num;
9       this.price=price;
10    }
11    public void display() {
12      System.out.println("苹果的数量: "+num);
13      System.out.println("苹果的单价: "+price);
```

```
14    }
15    public static void main(String args[]){
16      Apple a1=new Apple();
17      Apple a2=new Apple(50,3.15)
18      a1.display();
19      a2.display();
20    }
21  }
```

【运行结果】

```
E:\Java\JNBExamples>java Apple
苹果的数量：50
苹果的单价：3.15
```

【分析讨论】

- 在类 Apple 的定义中，第 4～6 行语句定义了 1 个没有参数的构造方法；第 7～10 行语句定义了含有 2 个参数的构造方法。
- 在 main()方法中，通过这两个构造方法分别创建了两个对象 a1、a2。其中，a1 的成员变量的值为 10 和 2；a2 的成员变量的值为传递的实际参数 50 和 3.15。然后，通过对象 a1 和 a2 调用 display()方法输出各自成员变量的值，得到上述输出结果。

2. 缺省的构造方法

在 Java 语言中，类可以不定义构造方法，而其他的类仍然可以通过调用无参数的构造方法实现该类的实例化。这是因为 Java 语言为每个类都自动提供一个特殊的、不带参数且方法体为空的缺省构造方法。其语法形式如下。

```
public <className>{ }
```

- 当用缺省的构造方法初始化对象时，系统将用默认值初始化对象的成员变量。
- 一旦在类中定义了显式的构造方法，无论一个亦或多个，系统都将不再提供缺省的无参数构造方法。此时如果在程序中使用缺省的构造方法，则会出现编译错误。

3. 构造方法的重载

构造方法也可以重载，重载的目的是使类的对象具有不同的初始值，从而为对象的初始化提供便利。一个类中的若干个构造方法之间还可以相互调用。当一个构造方法需要调用另一个构造方法时，可以使用关键字 this。同时，这条调用语句应该是整个构造方法的第一条可执行语句，这样可以最大限度地提高已有代码的利用率，减少维护程序的工作量。

【例 3.10】 对类 Apple 的构造方法进行重载，使用关键字 this 引用同类的其他构造方法。

```
1   class Apple {
2     private String color;
3     private int num;
4     public Apple(String c,int n) {
```

```
5       color=c;
6       num=n;          }
7     public Apple(String c) {
8       this(c,0);          }
9     public Apple() {
10      this("Unknown");   }
11    public String getColor() {
12      return color;         }
13    public int getNum() {
14      return num;     }
15  }
16  public class AppleDemo {
17    public static void main(String args[]) {
18      Apple apple=new Apple();
19      System.out.println("苹果颜色: "+apple.getColor());
20      System.out.println("苹果数量: "+apple.getNum());   }
21  }
```

【运行结果】

```
E:\Java\JNBExamples>java AppleDemo
苹果颜色: Unknown
苹果数量: 0
```

【分析讨论】

- 关键字 this 用来调用同类的其他构造方法。
- 在 main()方法中，第 18 行语句调用了无参数的构造方法 Apple()，它的执行导致第 10 行语句调用了含有 1 个参数的构造方法，而它的执行同样导致第 8 行语句调用了含有 2 个参数的构造方法(第 4 行语句)。

3.2　封装与数据隐藏

封装是 OOP 的一个重要特性。一般地，封装是将客户端不应看到的信息包裹起来，使内部的执行对外部来看是一种不透明的黑箱，客户端不需要了解内部资源就能够达到目的。为数据提供良好的封装是保证类设计的基本方法之一。

3.2.1　封 装

封装也称数据隐藏，是指将对象的数据与操作数据的方法相结合，通过方法将对象的数据与实现细节保护起来，只保留一些对外接口，以便与外界发生联系。系统的其他部分只有通过包裹在数据外面的被授权的操作访问对象，因此封装同时也实现了对象的隐藏。也就是说，用户无须知道对象的内部方法的实现细节，但可以根据对象提供的外部接口(对象名和参数)访问对象。封装具有如下特征：

- 在类的定义中，通过设置访问对象的属性及方法的权限，限制该类的对象及其他类的对象的使用范围。
- 提供一个接口来描述其他对象的使用方法。
- 其他对象不能直接修改对象所拥有的属性和方法。

封装反映了事物的相对独立性。封装在编程上的作用是使对象以外的部分不能随意存取对象的内部数据，从而有效地避免了外部错误对它的“交叉感染”。通过封装和数据隐藏机制，将一个对象相关的变量和方法封装为一个独立的软件体，单独进行实现与维护，并使对象能够在系统内部方便地进行传递，另外也保证了对象数据的一致性，并使程序易于维护。OOP 的封装单位是对象。类的概念本身也具有封装的含义，因为对象的特性是由类描述的。

3.2.2 访问控制

访问控制是通过在类的定义中使用权限修饰符实现的，以达到保护类的变量和方法的目的。Java 语言支持 4 种不同的访问权限。

- 私有的：用 private 修饰符指定。
- 保护的：用 protected 修饰符指定。
- 共有的：用 public 修饰符指定。
- 默认的，也称为 default 或 package：不适用于修饰符指定。

访问控制的对象有包、类、接口、类成员和构造方法。除了包的访问控制由系统决定外，其他的均通过访问控制符实现。访问控制符是一组限定类、接口、类成员是否可以被其他类访问的修饰符。其中，类和接口只有 public 和 default 两种。类成员和构造方法可以是 public、private、protected 和 default 4 种，见表 3.1。

表 3.1 Java 语言的类成员的 4 种访问控制权限及其可见性

访问空字符	可否直接访问			
	同一个类中	同一个包中	不同包中的子类	任何场合
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√

1. private

类中带有 private 修饰符的成员只能在类的内部使用，在其他类中不允许直接访问。一般地，把不想让外界访问的数据和方法声明为 private，这有利于数据的安全并保证数据的一致性，也符合编程中隐藏内部信息处理细节的基本原则。

构造方法也可以定义为 private。如果一个类的构造方法声明为 private，则其他类不能生成该类的实例对象。一个类不能访问其他类的对象的 private 成员，但是同一个类两个对象之间可以相互访问对方的 private 成员，这是因为访问控制是在类层次上（不同类

的所有实例对象),而不是在对象级别上(同一个类的特定实例)。

【例 3.11】　在类 Parent 中,通过成员方法 isEqualTo(),验证同一个类的对象之间可以访问其私有的成员。

```
1   class Parent {
2     private int privateVar;
3     public Parent(int p) {
4     privateVar=p;
5   }
6   boolean isEqualTo(Parent anotherParent) {
7     if(this.privateVar==anotherParent.privateVar)
8       return true;
9     else
10      return false;
11    }
12  }
13   public class PrivateDemo {
14    public static void main(String[] args) {
15      Parent p1=new Parent(20);
16      Parent p2=new Parent(40);
17      System.out.println(p1.isEqualTo(p2));
18    }
19  }
```

【运行结果】

```
E:\Java\JNBExamples>java PrivateDemo
false
```

【分析讨论】

- 在 Parent 类中定义一个方法 isEqualTo(),用于比较 Parent 类的当前对象的私有变量 privateVar 与另一个对象 anotherParent 的私有变量 privateVar 是否相等。如果相等,则返回 true,否则返回 false。
- 在测试类 PrivateDemo 中,虽然 p1 和 p2 同为 Parent 类的对象,但是它们的私有成员变量 p1.privateVar 和 p2.privateVar 的值却不同,分别为 20 和 40,因此,最后的输出结果为 false。
- 该程序说明访问控制是应用于类(class)层次或者类型(type)层次,而不是应用于对象层次。

2. default

如果一个类没有显式地设置成员的访问控制级别,则说明它使用的是默认的访问权限,称为 default 或 package。default 权限允许被这个类本身,或者相同包中的类访问其成员,这个访问级别假设在相同包中的类是相互信任的。对于构造方法,如果不加任何访问权限,也是 default 权限,则除这个类本身和同一个包中的类以外,其他类均不能生成该

类的对象实例。

【例 3.12】 在类 Parent 中定义具有 default 访问权限的变量和方法。变量和方法属于 p1 包，Child 类也属于 p1 包，所以 p1 包中的其他类也可以访问 Parent 类的 default 的成员变量和方法。

```
1  package p1;
2  class Parent {
3     int packageVar;
4     void packageMethod() {
5       System.out.println("I am packageMethod!");
6     }
7  }
```

下面是 Child 类的定义。

```
1  package p2;
2  class Child {
3    void accessMethod() {
4      Alpha a=new Alpha();
5      a.packageVar=10;       //合法的
6      a.packageMethod();     //合法的
7    }
8  }
```

3. protected

protected 类型的类成员可以被同一个类、同一个包中的其他类以及它的子类访问。因此，在允许类的子类和相同包中的类访问而禁止其他不相关的类访问时，可以使用 protected 修饰符。protected 修饰符将子类和相同包中的类看作一个家族，只允许家族成员之间相互访问，而禁止这个家族之外的类和对象涉足其中。

假设定义了 3 个类：Parent、Person 和 Child，Child 类是 Parent 的子类。Parent 类和 Person 类在包 p1 中，而 Child 类在包 p2 中。Parent 类的定义如下：

```
1  package p1;
2  class Parent {
3     protected int protectedVar;
4     protected void protectMethod() {
5       System.out.println("I am protectedMethod!");
6     }
7  }
```

因为 Person 类与 Parent 类属于同一个包中，所以其可以访问 Parent 对象的 protected 成员变量和方法。下面是 Person 类的定义。

```
1  package p1;
2  class Person {
```

```
3     void accessMethod() {
4       Parent p=new Parent();
5       p.protectedVar=100;
6       p.protectedMethod();
7     }
8   }
```

Child 类继承了 Parent 类，但是却在包 p2 中。Child 类的对象虽然可以访问 Parent 类的 protected 成员变量和方法，但是只能通过 Child 类的对象或者它的子类对象访问，不能通过 Parent 类的对象直接对这两个类的 protected 成员进行访问。因此，Child 类的方法 accessMethod()试图通过 Parent 类的对象 p 访问其变量 protectedVar 和方法 protectedMethod()是非法的，而通过 Child 类的对象 c 访问该变量和方法则是合法的。下面是 Child 类的定义。

```
1   package p2;
2   import p1.*;
3   class Child extends Parent {
4     void accessMethod(Parent p, Child c) {
5       p.protectedVar=10;          //非法的
6       c.protectedVar=10;          //合法的
7       p.protectedMethod();        //非法的
8       c.protectedMethod();        //合法的
9     }
10  }
```

【分析讨论】

如果 Child 类与 Parent 类属于同一个包，则上述非法语句就是合法的了。

4. public

带有 public 修饰符的成员可以被所有的类访问。如果构造方法的访问权限为 public，则所有的类都可以生成该类的实例对象。一般地，一个成员只有在被外部对象使用后不会产生不良结果时，才会被声明为 public。在类中的方法被定义为 public，表示该方法是这个类对外的接口，程序的其他部分可以通过调用它们达到与当前类交换信息、传递消息甚至影响当前类的目的，从而避免程序的其他部分直接操作这个类的数据。

3.2.3　package 与 import

用面向对象技术开发软件系统时，程序员需要定义许多类并使之共同工作，有些类可能需要在多处反复被使用。为了使这些类易于查找和使用，避免命名冲突和限定类的访问权限，程序员可以将一组相关的类与接口包裹在一起形成一个包。包是接口和类的集合(或称为容器)，它将一组类集中到一起。Java 语言通过包就可以方便地管理程序中的类。包(package)的优点主要体现在以下 3 方面。

- 编程人员可以很容易地确定包中的类是相关的，并且根据所需的功能找到相应的类。

- 防止类命名混乱。每个包都创建一个新的命名空间，因此不同包中的相同的类名不会冲突。
- 控制包中的类、接口、成员变量和方法的可见性。在包中，除声明为 private 的私有成员外，类中所有的成员都可以被同一个包中的其他类和方法访问。

1. package 语句

包的创建就是将 Java 程序文件中的接口与类纳入指定的包中。创建包可以通过在类和接口的 Java 程序中用 package 语句实现。package 语句的语法如下：

```
package pk1[.pk2[.pk3...]];
```

其中，“.”符号代表目录分隔符，pk1 是最外层的包，pk2、pk3 依次是内层的包。

创建一个包就是在当前文件夹下创建一个子文件夹，存放这个包中包含的所有类的 class 文件。Java 编译器把包对应于文件系统的目录进行管理，因此包可以嵌套使用，即一个包中可以含有类的定义，也可以含有子包，其嵌套层数没有限制。

【例 3.13】 用关键字 package 将类 Circle 打包到 com 下的 graphics 包中。

```
1  package com.graphics;
2  public class Circle {
3     final float PI=3.14159f;       //定义一个用于表示圆周率的常量 PI
4     public static void main(String[] args) {
5       System.out.println("画一个圆形!");
6     }
7  }
```

类 Circle 属于 com.graphics 包，所以该类的名字为 com.graphics.Circle。假设 Circle.java 保存在 E:\JNBExamples\ch03 中，而类 com.graphics.Circle 位于 C:\mypkg 中，那么编译和运行该类的步骤如下：

(1) 将 C:\mypkg 添加到 CLASSPATH 中。

(2) 在命令行窗口中将 E:\JNBExamples\ch03 作为当前目录，输入编译命令 javac -d c:\mypkg Circle.java，则在当前目录下将产生 Circle.class 类文件(javac 命令中的-d 选项用于指定所产生的类文件的路径)。

(3) 在命令行窗口中输入 java com.graphics.Circle，则会得到运行结果“画一个圆”。

【分析讨论】

- package 语句必须在 Java 程序的第一行，该行之前只能有空格和注释。每个 Java 程序中只能有一条 package 语句，一个类只能属于一个包。
- 如果没有 package 语句，则 Java 程序为无名包，此时 Java 程序保存在当前目录下。

2. import 语句

通常，一个类只能引用与它在同一个包中的类。如果要使用其他包中的 public 类，则可以用以下两种方式。

- 导入包中的类。在每个要导入的类的名称前加上完整的包名。
- 使用 import 语句导入包中的类。其语法如下：