第8章
Introduction to Java Programming, 8E

对象和类

学习目标

- 描述对象和类，使用类来建模对象（8.2节）。
- 使用UML图形符号来描述类和对象（8.2节）。
- 演示如何定义类以及如何创建对象（8.3节）。
- 使用构造方法创建对象（8.4节）。
- 通过对象引用变量访问对象（8.5节）。
- 使用引用类型定义引用变量（8.5.1节）。
- 使用对象成员访问操作符（.）来访问对象的数据和方法（8.5.2节）。
- 定义引用类型的数据域并给对象的数据域赋默认值（8.5.3节）。
- 区分对象引用变量和基本类型变量的不同（8.5.4节）。
- 使用Java类库中的`Data`类、`Random`类和`JFrame`类（8.6节）。
- 区分实例变量与静态变量、实例方法与静态方法的不同（8.7节）。
- 定义有恰当的`get`方法和`set`方法的私有数据域（8.8节）。
- 封装数据域以便于类的维护（8.9节）。
- 开发带对象参数的方法，区分基本类型参数和对象类型参数的不同（8.10节）。
- 在数组中存储和处理对象（8.11节）。

8.1 引言

学习过前几章的内容之后，你已经能够使用选择、循环、方法和数组解决很多程序设计问题。但是，这些Java的特性还不足够用来开发图形用户界面和大型软件系统。假设希望开发一个GUI（图形用户界面，发音为goo-ee），如图8-1所示，该如何用程序实现它呢？

图8-1 从类中创建这些GUI对象

本章开始介绍面向对象程序设计，它会有助于更有效地开发GUI和大型软件系统。

8.2 定义对象的类

面向对象程序设计（OOP）就是使用对象进行程序设计。*对象*（object）代表现实世界中可以明确标识的一个实体。例如：一个学生、一张桌子、一个圆、一个按钮甚至一笔贷款都可以看作是一个对象。每个对象都有自己独特的标识、状态和行为。

- 一个对象的状态（state，也称之为*特征*（property）或*属性*（attribute））是指那些具有它们当前值的*数据域*。例如：圆对象具有一个数据域`radius`，它是标识圆的属性。一个矩形对象具有数据域`width`和`height`，它们都是标识矩形的属性。
- 一个对象的行为（behavior，也称之为*动作*（action））是由方法定义的。调用对象的一个方法就是

要求对象完成一个动作。例如：可以为圆对象定义一个名为getArea()的方法。圆对象可以调用getArea()返回圆的面积。

使用一个通用类来定义同一类型的对象。类是一个模板、蓝本或者说是合约，用来定义对象的数据域是什么以及方法是做什么的。一个对象是类的一个实例。可以从一个类中创建多个实例。创建实例的过程称为实例化（instantiation）。术语对象（object）和实例（instance）经常是可以互换的。类和对象之间的关系类似于苹果派配方和苹果派之间的关系。可以用一种配方做出任意多的苹果派来。图8-2显示名为Circle的类和它的三个对象。

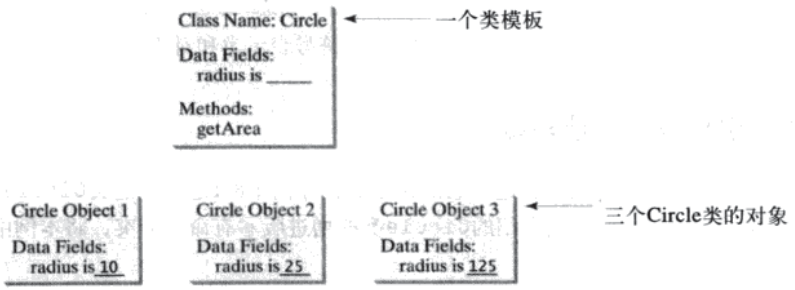

图8-2 类是创建对象的模板

Java类使用变量定义数据域，使用方法定义动作。除此之外，类还提供了一种称为构造方法（constructor）的特殊类型的方法，调用它可以创建一个新对象。构造方法本身是可以完成任何动作的，但是设计构造方法的初衷还是为了完成初始化动作，例如：初始化对象的数据域。图8-3显示定义圆对象的类的例子。

```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }
}
```


图8-3 类是一个定义相同类型对象的结构体

Circle类与目前所见过的所有其他类都不同，它没有main方法，因此是不能运行的；它只是对圆对象的定义。本书还将涉及包含main方法的类。为了方便，本书将包含main方法的类称为主类（main class）。

图8-2中类的模板和对象的图解可以使用统一建模语言（Unified Modeling Language，UML）的图形符号进行标准化。如图8-4所示，这种符号称为UML类图（UML class diagram），或简称为类图（class diagram）。在类图中，数据域表示为：

dataFieldName: dataFieldType 数据域名:数据域类型

构造方法可以表示为：

ClassName(parameterName: parameterType) 类名（参数名:参数类型）

方法可以表示为：

methodName(parameterName: parameterType): returnType 方法名（参数名:参数类型）:返回类型

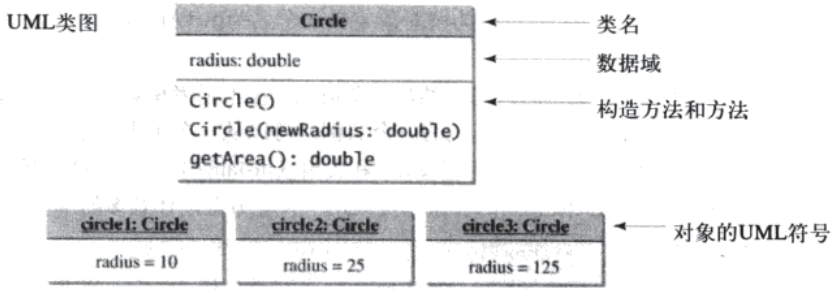

图8-4 可以使用UML符号表示类和对象

8.3 举例：定义类和创建对象

本节给出两个定义类和使用类创建对象的例子。程序清单8-1是一个定义Circle类并使用该类创建对象的程序。为了避免与本章后续介绍的Circle类的增进版本有命名冲突，将本例中的Circle类命名为Circle1。

程序构造了三个圆对象，其半径分别为1.0、25和125，然后显示这三个圆的半径和面积。将第二个对象的半径改为100，然后显示它的新半径和面积。

程序清单8-1 TestCircle1.java

```
1 public class TestCircle1 {
2   /** Main method */
3   public static void main(String[] args) {
4     // Create a circle with radius 1.0
5     Circle1 circle1 = new Circle1();
6     System.out.println("The area of the circle of radius "
7       + circle1.radius  + " is " + circle1.getArea() );
8
9     // Create a circle with radius 25
10    Circle1 circle2 = new Circle1(25);
11    System.out.println("The area of the circle of radius "
12      + circle2.radius + " is " + circle2.getArea());
13
14    // Create a circle with radius 125
15    Circle1 circle3 = new Circle1(125);
16    System.out.println("The area of the circle of radius "
17      + circle3.radius + " is " + circle3.getArea());
18
19    // Modify circle radius
20    circle2.radius = 100;
21    System.out.println("The area of the circle of radius "
22      + circle2.radius + " is " + circle2.getArea() );
23  }
24 }
25
26 // Define the circle class with two constructors
27 class Circle1 {
28   double radius ;
29
30   /** Construct a circle with radius 1 */
31   Circle1() {
32     radius = 1.0;
33   }
34
35   /** Construct a circle with a specified radius */
36   Circle1(double newRadius)  {
37     radius = newRadius;
38   }
```

```
39
40   /** Return the area of this circle */
41   double getArea() {
42     return radius * radius * Math.PI;
43   }
44 }
```

```
The area of the circle of radius 1.0 is 3.141592653589793
The area of the circle of radius 25.0 is 1963.4954084936207
The area of the circle of radius 125.0 is 49087.385212340516
The area of the circle of radius 100.0 is 31415.926535897932
```

程序包括两个类。其中第一个类TestCircle1是主类。它的唯一目的就是测试第二个类Circle1。使用这样的类的程序通常称为是该类的客户（client）。运行这个程序时，Java运行系统会调用这个主类的main方法。

可以把两个类放在同一个文件中，但是文件中只能有一个类是公共的。此外，公共类必须与文件同名。因此，文件名就应该是TestCircle1.java，因为TestCircle1是公共的。

主类包含main方法（第3行），该方法创建三个对象。和创建数组一样，使用new操作符从构造方法创建一个对象。new Circle1()创建一个半径为1.0的对象（第5行），new Circle1(25)创建一个半径为25的对象（第10行），而new Circle1(125)创建一个半径为125的对象（第15行）。

这三个对象（通过circle1、circle2和circle3来引用）有不同的数据，但是有相同的方法。因此，可以使用getArea()方法计算它们各自的面积。可以分别使用circle1.radius、circle2.radius、circle3.radius来通过对象引用访问数据域。对象可以分别使用circle1.getArea()、circle2.getArea()、circle3.getArea()来通过对象引用调用它的方法。

这三个对象是独立的。circle2的半径在第20行改为100。这个对象的新半径和新面积在第21～22行显示。

编写Java程序的方法有很多种。例如：可以将例子中的两个类组合成一个，如程序清单8-2所示。

程序清单8-2 Circle1.java

```
1 public class Circle1 {
2   /** Main method */
3   public static void main(String[] args) {
4     // Create a circle with radius 1.0
5     Circle1 circle1 = new Circle1();
6     System.out.println("The area of the circle of radius "
7       + circle1.radius + " is " + circle1.getArea());
8
9     // Create a circle with radius 25
10    Circle1 circle2 = new Circle1(25);
11    System.out.println("The area of the circle of radius "
12      + circle2.radius + " is " + circle2.getArea());
13
14    // Create a circle with radius 125
15    Circle1 circle3 = new Circle1(125);
16    System.out.println("The area of the circle of radius "
17      + circle3.radius + " is " + circle3.getArea());
18
19    // Modify circle radius
20    circle2.radius = 100;
21    System.out.println("The area of the circle of radius "
22      + circle2.radius + " is " + circle2.getArea());
23  }
24
25  double radius;
26
27  /** Construct a circle with radius 1 */
```

```
28   Circle1() {
29     radius = 1.0;
30   }
31
32   /** Construct a circle with a specified radius */
33   Circle1(double newRadius) {
34     radius = newRadius;
35   }
36
37   /** Return the area of this circle */
38   double getArea() {
39     return radius * radius * Math.PI;
40   }
41 }
```

由于组合的类中有一个main方法，所以它可以由Java解释器来执行。main方法和程序清单8-1中的是一样的。它演示如何通过在一个类中只要加入main方法，就能测试这个类。

另一个例子是关于电视机的。每台电视机都是一个对象，每个对象都有状态（当前频道、当前音量、电源开或关）以及动作（转换频道、调节音量、开启/关闭）。可以使用一个类对电视机进行建模。这个类的UML图如图8-5所示。

程序清单8-3给出定义TV类的程序。

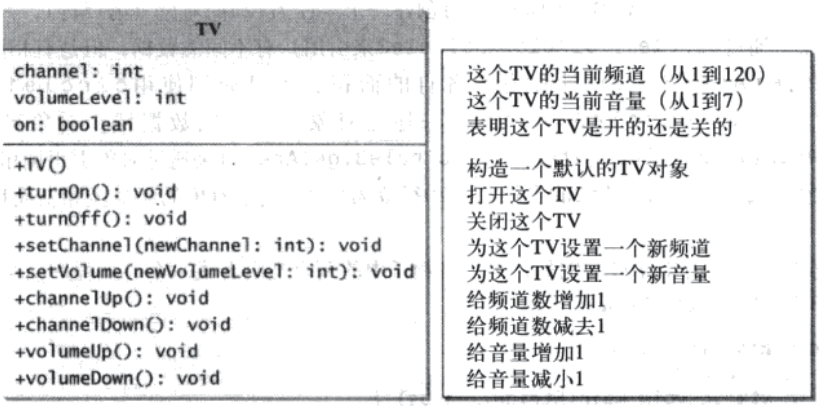

图8-5 TV类是对电视机的建模

程序清单8-3 TV.java

```
1 public class TV {
2   int channel = 1; // Default channel is 1
3   int volumeLevel = 1; // Default volume level is 1
4   boolean on = false; // By default TV is off
5
6   public TV() {
7   }
8
9   public void turnOn() {
10    on = true;
11  }
12
13  public void turnOff() {
14    on = false;
15  }
16
17  public void setChannel(int newChannel) {
18    if (on && newChannel >= 1 && newChannel <= 120)
19      channel = newChannel;
20  }
```

```
21
22   public void setVolume(int newVolumeLevel) {
23     if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
24       volumeLevel = newVolumeLevel;
25   }
26
27   public void channelUp() {
28     if (on && channel < 120)
29       channel++;
30   }
31
32   public void channelDown() {
33     if (on && channel > 1)
34       channel--;
35   }
36
37   public void volumeUp() {
38     if (on && volumeLevel < 7)
39       volumeLevel++;
40   }
41
42   public void volumeDown() {
43     if (on && volumeLevel > 1)
44       volumeLevel--;
45   }
46 }
```

注意　如果没有打开电视，那么频道和音量都没有改变。在改变它们中的任何一个之前，要检查它的当前值以确保它在一个正确的范围内。

程序清单8-4给出使用TV类创建两个对象的程序。

程序清单8-4　TestTV.java

```
1 public class TestTV {
2   public static void main(String[] args) {
3     TV tv1 = new TV();
4     tv1.turnOn();
5     tv1.setChannel(30);
6     tv1.setVolume(3);
7
8     TV tv2 = new TV();
9     tv2.turnOn();
10    tv2.channelUp();
11    tv2.channelUp();
12    tv2.volumeUp();
13
14    System.out.println("tv1's channel is " + tv1.channel
15      + " and volume level is " + tv1.volumeLevel );
16    System.out.println("tv2's channel is " + tv2.channel
17      + " and volume level is " + tv2.volumeLevel);
18  }
19 }
```

```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level is 2
```

程序在第3行和第8行创建两个对象，然后调用对象中的方法来完成设置频道和音量的动作，以及增加频道和提高音量的动作。程序在第14～17行显示对象的状态。使用像tv1.turnOn()的语法调用这个方法（第4行）。使用像tv1.channel的语法访问数据域（第14行）。

这些例子展示了类和对象的概貌。你可能已经有很多关于构造方法、对象、引用变量、访问数据域以及调用对象方法方面的问题，本节随后将会详细讨论这些话题。

8.4 使用构造方法构造对象

构造方法是一种特殊的方法。它们有以下三个特殊性：

1）构造方法必须具备和所在类相同的名字。

2）构造方法没有返回类型，甚至连void也没有。

3）构造方法是在创建一个对象使用new操作符时调用的。构造方法的作用是初始化对象。

构造方法具有和定义它的类完全相同的名字。和所有其他方法一样，构造方法也可以重载（也就是说，可以有多个同名的构造方法，但它们要有不同的签名），这样更易于用不同的初始数据值来构造对象。

一个常见的错误就是将关键字void放在构造方法的前面。例如：

```
public void Circle() {
}
```

在这种情况下，Circle()是一个方法，而不是构造方法。

构造方法是用来构造对象的。为了能够从一个类构造对象，使用new操作符调用这个类的构造方法，如下所示：

```
new ClassName(arguments); new 类名(参数);
```

例如：new Circle()使用Circle类中定义的第一个构造方法创建一个Circle对象。new Circle(25)调用Circle类中定义的第二个构造方法创建一个Circle对象。

通常，一个类会提供一个没有参数的构造方法（例如：Circle()）。这样的构造方法称为*无参构造方法*（no-arg或no-argument constructor）。

一个类可以不定义构造方法。在这种情况下，类中隐含定义一个方法体为空的无参构造方法。这个构造方法称为*默认构造方法*（default constructor），*当且仅当类中没有明确定义任何构造方法时才会自动提供它*。

8.5 通过引用变量访问对象

要给新创建的对象在内存中分配空间。它们可以通过引用变量来访问。

8.5.1 引用变量和引用类型

对象是通过对象*引用变量*（reference variable）来访问的，该变量包含对对象的引用，使用如下语法格式声明这样的变量：

```
ClassName objectRefVar; 类名 对象引用变量;
```

一个类基本上等同于一个程序员定义的类型。一个类就是一种*引用类型*（reference type），这意味着任何类型为类的变量都可以引用该类的一个实例。下面的语句声明变量myCircle的类型是Circle类型：

```
Circle myCircle;
```

变量myCircle能够引用一个Circle对象。下面的语句创建一个对象，并且将它的引用赋给变量myCircle：

```
myCircle = new Circle();
```

利用如下所示的语法，可以写一条包括声明对象引用变量、创建对象以及将对象的引用赋值给这个变量的语句。

```
ClassName objectRefVar = new ClassName(); 类名 对象引用变量 = new 类名();
```

下面是一个例子：

```
Circle myCircle = new Circle();
```

变量myCircle中放的是对Circle对象的一个引用。

注意 从表面上看，对象引用变量中似乎存放了一个对象，但事实上，它只是包括了对该对象的引用。严格地讲，对象引用变量和对象是不同的，但是大多数情况下，这种差异是可以忽略的。因此，可以简单地说myCircle是一个Circle对象，而不用冗长地描述说，myCircle是一个包含对Circle对象的引用变量。

注意 在Java中，数组被看作是对象。数组是用new操作符创建的。一个数组变量实际上是一个包含数组引用的变量。

8.5.2 访问对象的数据和方法

在创建一个对象之后，它的数据和方法可以使用圆点运算符（.）来访问和调用，该运算符也称为对象成员访问运算符（object member access operator）：

- objectRefVar.dataField引用对象的数据域。
- objectRefVar.method(参数)调用对象的方法。

例如：myCircle.radius引用myCircle的半径，而myCircle.getArea()调用myCircle的getArea方法。方法作为对象上的操作被调用。

数据域radius称作实例变量（instance variable），因为它依赖于某个具体的实例。基于同样的原因，getArea方法称为实例方法（instance method），因为只能在具体的实例上调用它。调用对象上的实例方法的过程称为调用对象（calling object）。

警告 回想一下，我们曾经使用过Math.methodName（参数）（例如：Math.pow(3,2.5)）来调用Math类中的方法。那么能否用Circle.getArea()来调用getArea方法呢？答案是不能。Math类中的所有方法都是用关键字static定义的静态方法。但是，getArea()是实例方法，因此它是非静态的。它必须使用objectRefVar.methodName（参数）的方式（例如：myCircle.getArea()）从对象调用。更详细的解释将在8.7节中给出。

注意 大多数时候，我们创建一个对象，然后会将它赋值给一个变量。接下来，就可以使用这个变量来引用对象。有时候，一个对象在创建之后并不需要引用。在这种情况下，可以创建一个对象，而并不将它明确地赋值给一个变量，如下所示：

```
new Circle();
```

或者

```
System.out.println("Area is " + new Circle(5).getArea());
```

前面的语句创建了一个Circle对象。后面的语句创建了一个Circle对象，然后调用它的getArea方法返回其面积。这种方式创建的对象称为匿名对象（anonymous object）。

8.5.3 引用数据域和null值

数据域也可能是引用型的。例如：下面的Student类包含一个String类型的name数据域，String是一个预定义的Java类。

```
class Student {
  String name; // name has default value null
  int age; // age has default value 0
  boolean isScienceMajor; // isScienceMajor has default value false
  char gender; // gender has default value '\u0000'
}
```

如果一个引用类型的数据域没有引用任何对象，那么这个数据域就有一个特殊的Java值null。null同true和false一样都是一个直接量。true和false是boolean类型直接量，而null是引用类型直接量。

引用类型数据域的默认值是null，数值类型数据域的默认值是0，boolean类型数据域的默认值是false，而char类型数据域的默认值是'\u0000'。但是，Java没有给方法中的局部变量赋默认值。下面的代码显示Student对象中数据域name、age、isScienceMajor和gender的默认值：

```
class Test {
  public static void main(String[] args) {
    Student student = new Student();
    System.out.println("name? " + student.name);
    System.out.println("age? " + student.age);
    System.out.println("isScienceMajor? " + student.isScienceMajor);
    System.out.println("gender? " + student.gender);
  }
}
```

下面代码中的局部变量x和y都没有被初始化，所以它会出现编译错误：

```
class Test {
  public static void main(String[] args) {
    int x; // x has no default value
    String y; // y has no default value
    System.out.println("x is " + x);
    System.out.println("y is " + y);
  }
}
```

警告 NullPointerException是一种常见的运行错误，当调用值为null的引用变量上的方法时会发生此类异常。在通过引用变量调用一个方法之前，确保先将对象引用赋值给这个变量。

8.5.4 基本类型变量和引用类型变量的区别

每个变量都代表一个存储值的内存位置。声明一个变量时，就是在告诉编译器这个变量可以存放什么类型的值。对基本类型变量来说，对应内存所存储的值是基本类型值。对引用类型变量来说，对应内存所存储的值是一个引用，是对象的存储地址。例如：如图8-6所示，int型变量i的值就是int值1，而Circle对象c的值存的是一个引用，它指明这个Circle对象的内容存储在内存中的什么位置。

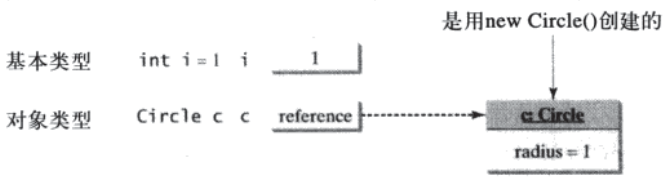

图8-6 基本类型变量在内存中存储的是一个基本类型值，而引用类型变量存储的是一个引用，它指向对象在内存中的位置

将一个变量赋值给另一个变量时，另一个变量就被赋了同样的值。对基本类型变量而言，就是将一个变量的实际值赋给另一个变量。对引用类型变量而言，就是将一个变量的引用赋给另一个变量。如图8-7所示，赋值语句i=j将基本类型变量j的内容复制给基本类型变量i。如图8-8所示，对引用变量来讲，赋值语句c1=c2是将c2的引用赋给c1。赋值之后，变量c1和c2指向同一个对象。

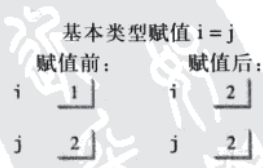

图8-7 基本类型变量j复制到变量i中

注意 如图8-8所示，执行完赋值语句c1=c2之后，c1指向c2所指的同一个对象。c1以前引用的对象就不再有用，因此，现在它就成为垃圾（garbage）。垃圾会占用内存空间。Java运行系统会检测垃圾并自动回收它所占的空间，这个过程称为垃圾回收（garbage collection）。

提示 如果你认为不再需要某个对象，可以显式地给该对象的引用变量赋null值。如果该对象

没有被任何引用变量所引用，Java虚拟机将自动回收它所占的空间。

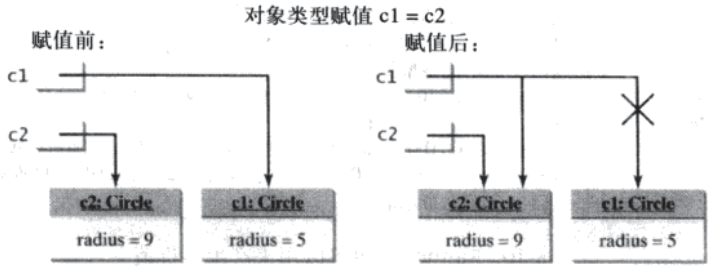

图8-8　引用变量c2复制到变量c1中

8.6　使用Java库中的类

程序清单8-1声明了Circle1类，并从这个类创建了该类的对象。你将会频繁地使用到Java类库里的类来开发程序。本节将给出Java类库中一些类的例子。

8.6.1　Date类

在程序清单2-6中，已经学习了如何使用System.currentTimeMillis()来获得当前时间。使用除法和求余运算分解出当前的秒数、分钟数和小时数。Java在java.util.Date类中还提供了与系统无关的对日期和时间的封装，如图8-9所示。

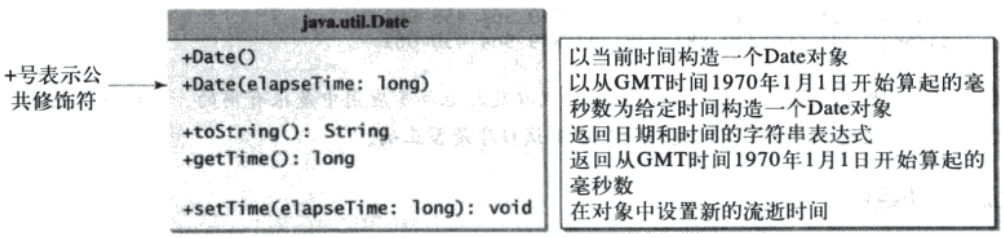

图8-9　Date对象表示特定的日期和时间

可以使用Date类中的无参构造方法为当前的日期和时间创建一个实例，它的getTime()方法返回自从GMT时间1970年1月1日算起至今逝去的时间，它的toString方法返回日期和时间的字符串。例如，下面的代码

```
java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");
System.out.println(date.toString());
```

显示输出为：

```
The elapsed time since Jan 1, 1970 is 1100547210284 milliseconds
Mon Nov 15 14:33:30 EST 2004
```

Date类还有另外一个构造方法：Date(long elapseTime)，可以用它创建一个Date对象。该对象有一个从GMT时间1970年1月1日算起至今逝去的以毫秒为单位的给定时间。

8.6.2　Random类

可以使用Math.random()获取一个0.0到1.0（不包括1.0）之间的随机double型值。另一种产生随机数的方法是使用如图8-10所示的java.util.Random类，它可以产生一个int、long、double、float和boolean型值。

java.util.Random	
+Random()	以当前时间作为种子构造Random对象
+Random(seed: long)	以特定种子构造Random对象
+nextInt(): int	返回一个随机整型值
+nextInt(n: int): int	返回一个在0到n之间（不包括0和n）的随机整型值
+nextLong(): long	返回一个随机的long型值
+nextDouble(): double	返回一个在0.0到1.0之间（不包括0.0和1.0）的随机double型值
+nextFloat(): float	返回一个在0.0F到1.0F之间（不包括0.0F和1.0F）的随机float型值
+nextBoolean(): boolean	返回一个随机布尔值

图8-10 Random对象可以用来产生随机值

创建一个Random对象时，必须指定一个种子或者使用默认的种子。无参构造方法使用当前已经逝去的时间作为种子，创建一个Random对象。如果这两个Random对象有相同的种子，那它们将产生相同的数列。例如：下面的代码都用相同的种子3来产生两个Random对象。

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
  System.out.print(random1.nextInt(1000) + " ");

Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
  System.out.print(random2.nextInt(1000) + " ");
```

这些代码产生相同的int类型的随机数列：

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

注意 产生相同随机值序列的能力在软件测试以及其他许多应用中是很有用的。可以在使用不同随机数序列之前，使用固定的随机序列以测试程序是否正确。

8.6.3 显示GUI组件

教学注意 图形用户界面（GUI）组件是讲授OOP的很好的例子。为了教授OOP，先介绍一些简单的GUI的例子。对GUI程序设计的完整介绍从第12章开始。

开发程序创建图形用户界面时，将会用到像JFrame、JButton、JRadioButton、JComboBox和JList这样的Java类来创建框架、单选按钮、组合框、列表等。程序清单8-5是一个使用JFrame类创建两个窗口的例子。这个程序的输出如图8-11所示。

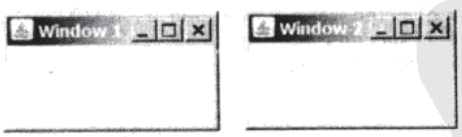

图8-11 这个程序使用JFrame类创建两个窗口

程序清单8-5 TestFrame.java

```
1 import javax.swing.JFrame;
2
3 public class TestFrame {
4   public static void main(String[] args) {
5     JFrame frame1 = new JFrame();
6     frame1.setTitle("Window 1");
7     frame1.setSize(200, 150);
8     frame1.setLocation(200, 100);
9     frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
10      frame1.setVisible(true);
11
12      JFrame frame2 = new JFrame();
13      frame2.setTitle("Window 2");
14      frame2.setSize(200, 150);
15      frame2.setLocation(410, 100);
16      frame2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17      frame2.setVisible(true);
18    }
19 }
```

这个程序创建两个JFrame类的对象（第5、12行），然后使用方法setTitle、setSize、setLocation、setDefaultCloseOperation和setVisible来设置这两个对象的属性。方法setTitle设置窗口的标题（第6、13行）。方法setSize设置窗口的宽度和高度（第7、14行）。方法setLocation指明窗口左上角的位置（第8、15行）。方法setDefaultCloseOperation在关闭框架时终止该程序（第9、16行）。方法setVisible表示是否显示这个窗口。

可以给窗口添加像按钮、标签、文本域、复选框和组合框这样的组件。组件是使用类来定义的。程序清单8-6给出一个创建图形用户界面的例子，如图8-1所示。

程序清单8-6 GUIComponents.java

```
 1 import javax.swing.*;
 2
 3 public class GUIComponents {
 4   public static void main(String[] args) {
 5     // Create a button with text OK
 6     JButton jbtOK = new JButton("OK");
 7
 8     // Create a button with text Cancel
 9     JButton jbtCancel = new JButton("Cancel");
10
11     // Create a label with text "Enter your name: "
12     JLabel jlblName = new JLabel("Enter your name: ");
13
14     // Create a text field with text "Type Name Here"
15     JTextField jtfName = new JTextField("Type Name Here");
16
17     // Create a check box with text bold
18     JCheckBox jchkBold = new JCheckBox("Bold");
19
20     // Create a check box with text italic
21     JCheckBox jchkItalic = new JCheckBox("Italic");
22
23     // Create a radio button with text red
24     JRadioButton jrbRed = new JRadioButton("Red");
25
26     // Create a radio button with text yellow
27     JRadioButton jrbYellow = new JRadioButton("Yellow");
28
29     // Create a combo box with several choices
30     JComboBox jcboColor = new JComboBox(new String[]{"Freshman",
31       "Sophomore", "Junior", "Senior"});
32
33     // Create a panel to group components
34     JPanel panel = new JPanel();
35     panel.add(jbtOK); // Add the OK button to the panel
36     panel.add(jbtCancel); // Add the Cancel button to the panel
37     panel.add(jlblName); // Add the label to the panel
38     panel.add(jtfName); // Add the text field to the panel
39     panel.add(jchkBold); // Add the check box to the panel
40     panel.add(jchkItalic); // Add the check box to the panel
41     panel.add(jrbRed); // Add the radio button to the panel
42     panel.add(jrbYellow); // Add the radio button to the panel
```

```
43      panel.add(jcboColor); // Add the combo box to the panel
44
45      JFrame frame = new JFrame(); // Create a frame
46      frame.add(panel); // Add the panel to the frame
47      frame.setTitle("Show GUI Components");
48      frame.setSize(450, 100);
49      frame.setLocation(200, 100);
50      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51      frame.setVisible(true);
52    }
53 }
```

这个程序使用类JButton、JLabel、JTextField、JCheckBox、JRadioButton和JComboBox来创建GUI对象（第6～31行）。然后，它使用JPanel类创建一个面板对象（第34行），并将按钮、标签、文本域、复选框、单选框以及组合框添加到这个面板中（第35～43行）。接着，程序创建一个框架，将刚才的面板添加到这个框架中（第45行）。第51行显示这个框架。

8.7 静态变量、常量和方法

程序清单8-1中Circle类的数据域radius称为一个*实例变量*。实例变量是绑定到类的某个特定实例的，它是不能被同一个类的不同对象所共享的。例如，假设创建了如下的两个对象：

```
Circle circle1 = new Circle();
Circle circle2 = new Circle(5);
```

circle1中的radius和circle2中的radius是不相关的，它们存储在不同的内存位置。circle1中radius的变化不会影响circle2中的radius，反之亦然。

如果想让一个类的所有实例共享数据，就要使用*静态变量*（static variable），也称之为*类变量*（class variable）。静态变量将变量值存储在一个公共的内存地址。因为它是公共的地址，所以如果某一个对象修改了静态变量的值，那么同一个类的所有对象都会受到影响。Java支持静态方法和静态变量，无须创建类的实例就可以调用*静态方法*（static method）。

修改Circle类，添加静态变量numberOfObjects统计创建的Circle对象的个数。当该类的第一个对象创建后，numberOfObjects的值是1。当第二个对象创建后，numberOfObjects的值是2。新Circle类的UML图如图8-12所示。Circle类定义了实例变量radius和静态变量numberOfObjects，还定义了实例方法getRadius、setRadius和getArea以及静态方法getNumberOfObjects。（注意，在UML类图中，静态变量和方法都是以下划线标注的。）

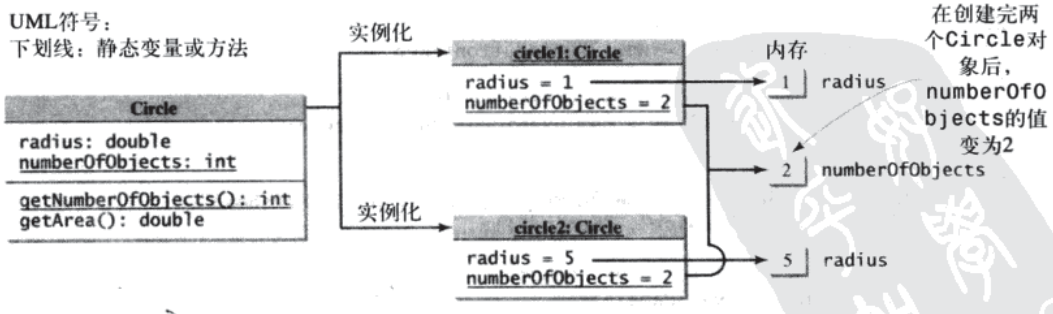

图8-12 属于实例的实例变量存储在互不相关的内存中。静态变量是被同一个类的所有实例所共享的

要声明一个静态变量或定义一个静态方法，就要在这个变量或方法的声明中加上修饰符static。静态变量numberOfObjects和静态方法getNumberOfObjects()可以如下声明：

```
static int numberOfObjects;

static int getNumberObjects() {
```

```
  return numberOfObjects;
}
```

类中的常量是被该类的所有对象所共享的。因此，常量应该声明为final static。例如：Math类中的常量PI是如下定义的：

```
final static double PI = 3.14159265358979323846;
```

新的名为Circle2的圆类的声明如程序清单8-7所示。

程序清单8-7 Circle2.java

```
1 public class Circle2 {
2   /** The radius of the circle */
3   double radius;
4
5   /** The number of objects created */
6   static int numberOfObjects = 0;
7
8   /** Construct a circle with radius 1 */
9   Circle2() {
10    radius = 1.0;
11    numberOfObjects++;
12  }
13
14  /** Construct a circle with a specified radius */
15  Circle2(double newRadius) {
16    radius = newRadius;
17    numberOfObjects++;
18  }
19
20  /** Return numberOfObjects */
21  static int getNumberOfObjects() {
22    return numberOfObjects;
23  }
24
25  /** Return the area of this circle */
26  double getArea() {
27    return radius * radius * Math.PI;
28  }
29 }
```

Circle2类中的getNumberOfObjects()方法是一个静态方法。静态方法的其他例子有JOptionPane类中的showMessageDialog方法和showInputDialog方法，以及Math类中所有的方法。main方法也是静态方法。

实例方法（例如：getArea()）和实例数据（例如：radius）都是属于实例的，所以它们在实例创建之后才能使用。它们是通过引用变量来访问的。静态方法（例如：getNumberOfObjects()）和静态数据（例如：numberOfObjects）可以通过引用变量或它们的类名来调用。

程序清单8-8中的程序演示如何使用实例变量、静态变量、实例方法和静态方法，还演示了使用它们的效果。

程序清单8-8 TestCircle2.java

```
1 public class TestCircle2 {
2   /** Main method */
3   public static void main(String[] args) {
4     System.out.println("Before creating objects");
5     System.out.println("The number of Circle objects is " +
6       Circle2.numberOfObjects);
7
8     // Create c1
9     Circle2 c1 = new Circle2();
10
11    // Display c1 BEFORE c2 is created
```

```
12      System.out.println("\nAfter creating c1");
13      System.out.println("c1: radius (" + c1.radius +
14        ") and number of Circle objects (" +
15        c1.numberOfObjects + ")");
16
17      // Create c2
18      Circle2 c2 = new Circle2(5);
19
20      // Modify c1
21      c1.radius = 9;
22
23      // Display c1 and c2 AFTER c2 was created
24      System.out.println("\nAfter creating c2 and modifying c1");
25      System.out.println("c1: radius (" + c1.radius +
26        ") and number of Circle objects (" +
27        c1.numberOfObjects + ")");
28      System.out.println("c2: radius (" + c2.radius +
29        ") and number of Circle objects (" +
30        c2.numberOfObjects + ")");
31    }
32 }
```

```
Before creating objects
The number of Circle objects is 0

After creating c1
c1: radius (1.0) and number of Circle objects (1)

After creating c2 and modifying c1
c1: radius (9.0) and number of Circle objects (2)
c2: radius (5.0) and number of Circle objects (2)
```

编译TestCircle2.java时，如果最后一次修改了Circle2.java之后还没有编译它，Java编译器就会自动编译Circle2.java。

静态变量和方法可以在不创建对象的情况下访问。第6行显示对象的个数为0，因为还没有创建任何对象。

main方法创建两个圆，c1和c2（第9、18行）。c1中的实例变量radius修改为9（第21行）。这个变化不会影响c2中的实例变量radius，因为这两个实例变量是独立的。c1创建之后静态变量numberOfObjects变成1（第9行），而c2创建之后numberOfObjects变成2（第18行）。

注意，PI是一个定义在Math中的常量，使用Math.PI来访问这个常量。可以用Circle2.numberOfObjects来代替c.numberOfObjects。这样可以提高可读性，因为读者可以很容易地识别静态变量。也可以用Circle2.getNumberOfObjects()替换Circle2.numberOfObjects。

提示 使用“类名.方法名（参数）”的方式调用静态方法，使用”类名.静态变量”的方式访问静态变量。这会提高可读性，因为读者可以很容易地识别出类中的静态方法和数据。

静态变量和静态方法既可以在类的实例方法中使用，也可以在类的静态方法中使用。但是，实例变量和实例方法只能在实例方法中使用，不能在静态方法中使用，因为静态变量和静态方法不属于某个特定的对象。因此，下面给出的代码就是错误的。

```
1 public class Foo {
2   int i = 5;
3   static int k = 2;
4
5   public static void main(String[] args) {
6     int j = i; // Wrong because i is an instance variable
7     m1(); // Wrong because m1() is an instance method
8   }
9
10  public void m1() {
```

```
11     // Correct since instance and static variables and methods
12     // can be used in an instance method
13     i = i + k + m2(i, k);
14   }
15
16   public static int m2(int i, int j) {
17     return (int)(Math.pow(i, j));
18   }
19 }
```

注意 如果用下面的代码替换第5～8行的代码，程序就是正确的，因为实例数据域i和方法m1是通过对象foo访问的（第6～7行）：

```
1 public class Foo {
2   int i = 5;
3   static int k = 2;
4
5   public static void main(String[] args) {
6     Foo foo = new Foo();
7     int j = foo.i; // OK, foo.i accesses the object's instance variable
8     foo.m1();   // OK. Foo.m1() invokes object's instance method
9   }
10
11  public void m1() {
12    i = i + k + m2(i, k);
13  }
14
15  public static int m2(int i, int j) {
16    return (int)(Math.pow(i, j));
17  }
18 }
```

设计指南 如何判断一个变量或方法应该是实例的还是静态的？如果一个变量或方法依赖于类的某个具体实例，那就应该将它定义为实例变量或实例方法。如果一个变量或方法不依赖于类的某个具体实例，就应该将它定义为静态变量或静态方法。例如：每个圆都有自己的半径。半径都依赖于某个具体的圆。因此，半径radius就是Circle类的一个实例变量。由于getArea方法依赖于某个具体的圆，所以，它也是一个实例方法。在Math类中没有一个方法是依赖于一个特定实例的，例如：random、pow、sin和cos。因此，这些方法都是静态方法。main方法也是静态的，可以从类中直接调用。

警告 一个常见的设计错误就是将一个本应该声明为静态的方法声明为实例方法。例如：方法factorial(int n)应该定义为静态的，如下所示，因为它不依赖于任何具体的实例。

```
public class Test {
  public int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
      result *= i;

    return result;
  }
}
```

a）错误的设计

```
public class Test {
  public static int factorial(int n)
    int result = 1;
    for (int i = 1; i <= n; i++)
      result *= i;

    return result;
  }
}
```

b）正确的设计

8.8 可见性修饰符

可以在类、方法和数据域前使用public修饰符，表示它们可以被任何其他的类访问。如果没有使用可见性修饰符，那么默认为类、方法和数据域是可以被同一个包中的任何一个类访问的。这称作包私有（package-private）或包内访问（package-access）。

注意 包可以用来组织类。为了完成这个目标，需要在程序中首先出现下面这行语句，在这行语句之前不能有注释也不能有空白：

```
package packageName;
```

如果定义类时没有声明包，就表示把它放在默认包中。

Java建议最好将类放入包中，而不要使用默认包。但是，本书为了简化问题，使用的是默认包。关于包的更多的信息，参见补充材料III.G。

除了public和默认可见性修饰符，Java还为类成员提供private和protected修饰符。本节介绍private修饰符。修饰符protected将在11.13节中介绍。

private修饰符限定方法和数据域只能在它自己的类中被访问。图8-13演示类C1中的公共的、默认的和私有的数据域或方法能否被同一个包内的类C2访问，以及能否被不在同一个包内的类C3访问。

```
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

图8-13 私有的修饰符将访问权限限定在它自己的类内，
默认修饰符将访问权限限定在包内，而公共的修饰符没有限定权限

如果一个类没有被定义为公共的，那么它只能在同一个包内被访问。如图8-14所示，C2可以访问C1，而C3不能访问C1。

```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

public class C3 {
  cannot access C1;
  can access C2;
}
```

图8-14 一个非公共类具有包访问性

可见性修饰符指明类中的数据域和方法是否能从该类之外被访问。在该类之内，对数据域和方法的访问是没有任何限制的。如图8-15b所示，Foo类的对象foo不能引用它的私有成员，因为foo在Test类中。如图8-15a所示，Foo类的对象foo可以访问它的私有成员，因为foo在自己的类内定义。

```
public class Foo {
  private boolean x;

  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert());
  }

  private int convert() {
    return x ? 1 : 1;
  }
}
```

a）因为对象foo是在Foo类中使用的，所以这样做是可以的

```
public class Test {
  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
    System.out.println(foo.convert());
  }
}
```

b）因为x和convert在Foo中是私有的，所以这样做是错误的

图8-15 如果一个对象是在它自己的类中定义的，那么这个对象可以访问它的私有成员

警告 修饰符private只能应用在类的成员上。修饰符public可以应用在类或类的成员上。在局部变量上使用修饰符public和private都会导致编译错误。

注意 大多数情况下，构造方法应该是公共的。但是，如果想防止用户创建类的实例，就该使用私有构造方法。例如：因为Math类的所有数据域和方法都是静态的，所以没必要创建Math类的实例。为了防止用户从Math类创建对象，在java.lang.Math中的构造方法定义为如下所示：

```
private Math() {
}
```

8.9 数据域封装

在程序清单8-7中，Circle2类的数据域radius和numberOfObjects可以直接修改（例如：myCircle.radius = 5或Circle2.numberOfObjects = 10）。这不是一个好的实例，原因有两点：

第一，数据可能被篡改。例如：numberOfObjects是用来统计被创建的对象的个数的，但是它可能会被错误地设置为一个任意值（例如：Circle2.numberOfObjects = 10）。

第二，它使类变得难于维护，同时容易出现错误。假如在其他程序已经使用Circle2类之后，想修改半径，以确保半径是一个非负数。因为使用该类的客户可以直接修改radius（例如：myCircle.radius=-5），所以，不仅要修改Circle2类，而且还要修改使用Circle2的这些程序。

为了避免对数据域的直接修改，应该使用private修饰符将数据域声明为私有的。这就称为数据域封装（data field encapsulation）。

在定义私有数据域的类外的对象是不能访问这个数据域的。但是经常会有客户端需要存取、修改数据域的情况。为了能够访问私有数据域，可以提供一个get方法返回数据域的值。为了能够更新一个数据域，可以提供一个set方法给数据域设置新值。

注意 通俗地讲，get方法称为读取器（getter）（或访问器（accessor）），而set方法称为设置器（setter）（或修改器（mutator））。

get方法有如下签名：

```
public returnType getPropertyName()
```

如果返回类型是boolean型，习惯上如下定义get方法：

```
public boolean isPropertyName()
```

set方法有如下签名：

```
public void setPropertyName(dataType propertyValue)
```

我们现在来创建一个新的圆类，半径设置为私有数据域，并有相关的访问器和修改器。类图如图8-16所示。程序清单8-9中定义一个名为Circle3的新圆的类。

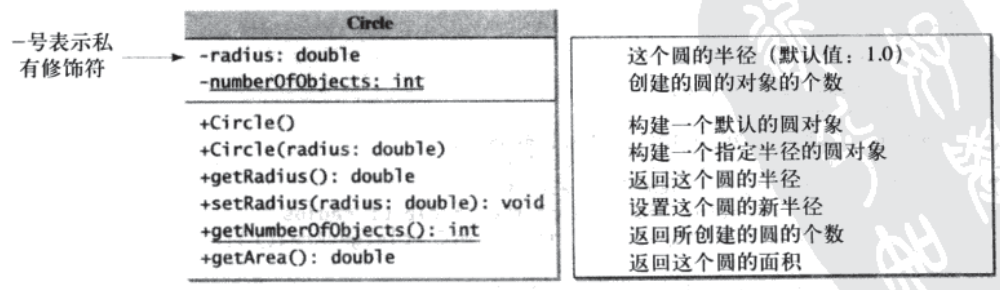

图8-16 Circle类封装了圆的属性并提供了get/set方法以及其他方法

程序清单8-9 Circle3.java

```
1 public class Circle3 {
2   /** The radius of the circle */
3   private double radius = 1;
4
5   /** The number of the objects created */
6   private static int numberOfObjects = 0;
7
8   /** Construct a circle with radius 1 */
9   public Circle3() {
10    numberOfObjects++;
11  }
12
13  /** Construct a circle with a specified radius */
14  public Circle3(double newRadius) {
15    radius = newRadius;
16    numberOfObjects++;
17  }
18
19  /** Return radius */
20  public double getRadius()  {
21    return radius;
22  }
23
24  /** Set a new radius */
25  public void setRadius(double newRadius) {
26    radius = (newRadius >= 0) ? newRadius : 0;
27  }
28
29  /** Return numberOfObjects */
30  public static int getNumberOfObjects() {
31    return numberOfObjects;
32  }
33
34  /** Return the area of this circle */
35  public double getArea() {
36    return radius * radius * Math.PI;
37  }
38 }
```

getRadius()方法（第20～22行）返回半径值，setRadius(newRadius)方法（第25～27行）给对象设置新的半径，如果新半径为负，就将这个对象的半径设置为0。因为这些方法是读取和修改半径的唯一途径，所以，你完全控制了如何访问radius属性。如果必须改变这些方法的实现，是不需要改变使用它们的客户程序的。这会使类更易于维护。

程序清单8-10给出一个客户程序，它使用Circle类创建一个Circle对象，然后使用setRadius方法修改它的半径。

程序清单8-10 TestCircle3.java

```
1 public class TestCircle3 {
2   /** Main method */
3   public static void main(String[] args) {
4     // Create a Circle with radius 5.0
5     Circle3 myCircle = new Circle3(5.0);
6     System.out.println("The area of the circle of radius "
7       + myCircle.getRadius() + " is " + myCircle.getArea());
8
9     // Increase myCircle's radius by 10%
10    myCircle.setRadius(myCircle.getRadius() * 1.1);
11    System.out.println("The area of the circle of radius "
12      + myCircle.getRadius() + " is " + myCircle.getArea());
13
14    System.out.println("The number of objects created is "
```

```
15        + Circle3.getNumberOfObjects() );
16    }
17 }
```

数据域radius被声明为私有的。私有数据只能在定义它们的类中被访问。不能在客户程序中使用myCircle.radius。如果企图从客户程序访问私有数据，将会产生编译错误。

由于numberOfObjects是私有的，所以它是不能修改的。这就制止了篡改行为。例如：用户不能设置numberOfObjects为100。要使这个值为100的唯一方法就是创建100个Circle类的对象。

假如通过把TestCircle类中的main方法移到Circle类中，实现将TestCircle类和Circle类组合成一个类，那么可以在main方法中使用myCircle.radius吗？参见复习题8.15找到这个答案。

设计指南 为防止数据被篡改以及使类更易于维护，最好将数据域声明为私有的。

8.10 给方法传递对象参数

可以将对象传递给方法。同传递数组一样，传递对象实际上是传递对象的引用。下面的代码将myCircle对象作为参数传递给printCircle方法：

```
1 public class Test {
2   public static void main(String[] args) {
3     // Circle3 is defined in Listing 8.9
4     Circle3 myCircle = new Circle3(5.0);
5     printCircle(myCircle);
6   }
7
8   public static void printCircle(Circle3 c) {
9     System.out.println("The area of the circle of radius "
10      + c.getRadius() + " is " + c.getArea());
11  }
12 }
```

Java只有一种参数传递方式：值传递（pass by value）。在上面的代码中，myCircle的值被传递给printCircle方法。这个值就是一个对Circle对象的引用值。

通过程序清单8-11中的程序，让我们来看一看传递基本类型值和传递引用值的差异。

程序清单8-11 TestPassObject.java

```
1 public class TestPassObject {
2   /** Main method */
3   public static void main(String[] args) {
4     // Create a Circle object with radius 1
5     Circle3 myCircle = new Circle3(1);
6
7     // Print areas for radius 1, 2, 3, 4, and 5.
8     int n = 5;
9     printAreas(myCircle, n);
10
11    // See myCircle.radius and times
12    System.out.println("\n" + "Radius is " + myCircle.getRadius());
13    System.out.println("n is " + n);
14  }
15
16  /** Print a table of areas for radius */
17  public static void printAreas(Circle3 c, int times) {
18    System.out.println("Radius \t\tArea");
19    while (times >= 1) {
20      System.out.println(c.getRadius() + "\t\t" + c.getArea());
21      c.setRadius(c.getRadius() + 1);
22      times--;
23    }
24  }
25 }
```

```
Radius                          Area
 1.0                             3.141592653589793
 2.0                            12.566370614359172
 3.0                            29.274333882308138
 4.0                            50.26548245743669
 5.0                            79.53981633974483
Radius is 6.0
n is 5
```

Circle3类是在程序清单8-9中定义的。这个程序使用Circle3类的一个对象myCircle和n的整数值调用printAreas(myCicle,n)方法（第9行），打印出半径为1、2、3、4和5的圆面积所构成的表格，如样本输出所示。

图8-17展示执行程序的这个方法的过程中对栈的调用。注意，对象是存储在堆中的。

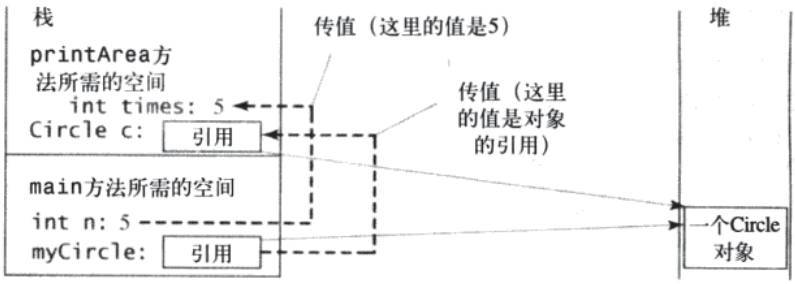

图8-17 n的值被传递给times，而myCircle的引用值被传递给printAreas方法中的c

当传递基本数据类型参数时，传递的是实参的值。在这种情况下，n(5)的值就被传递给times。在printAreas方法内，times的内容改变，这并不会影响n的内容。

传递引用类型的参数时，传递的是对象的引用。在这种情况下，c具有与myCircle相同的引用值。因此，通过在printAreas方法内部的c与在方法外的myCircle来改变对象的属性，效果是一样的。引用上的传值在语义上最好描述为传共享（pass-by-sharing）；也就是说，在方法中引用的对象和传递的对象是一样的。

8.11 对象数组

在第6章中创建的是基本类型元素的数组。也可以创建对象数组。例如，下面的语句声明并创建了10个Circle对象的数组：

```
Circle[] circleArray = new Circle[10];
```

为了初始化数组circleArray，可以使用如下的for循环：

```
for (int i = 0; i < circleArray.length; i++) {
  circleArray[i] = new Circle();
}
```

对象的数组实际上是引用变量的数组。因此，调用circleArray[1].getArea()实际上调用了两个层次的引用，如图8-18所示。circleArray引用了整个数组，circleArray[1]引用了一个Circle对象。

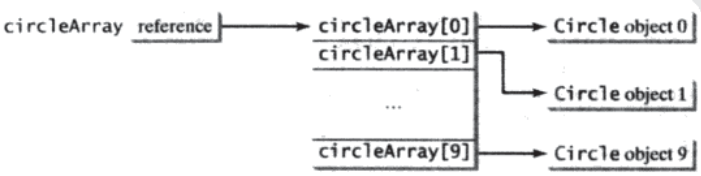

图8-18 在对象数组中，数组的每个元素都包含一个对象的引用

注意 当使用new操作符创建对象数组后，这个数组中的每个元素都是默认值为null的引用变量。

程序清单8-12给出了一个例子，演示如何使用对象数组。这个程序求圆的数组的总面积。程序创建5个Circle对象组成的数组circleArray，接着使用随机值初始化这些圆的半径，然后显示数组中的圆的总面积。

程序清单8-12 TotalArea.java

```
1  public class TotalArea {
2    /** Main method */
3    public static void main(String[] args) {
4      // Declare circleArray
5      Circle3[] circleArray;
6
7      // Create circleArray
8      circleArray = createCircleArray();
9
10     // Print circleArray and total areas of the circles
11     printCircleArray(circleArray);
12   }
13
14   /** Create an array of Circle objects */
15   public static Circle3[] createCircleArray() {
16     Circle3[] circleArray = new Circle3[5];
17
18     for (int i = 0; i < circleArray.length; i++) {
19       circleArray[i] = new Circle3(Math.random() * 100);
20     }
21
22     // Return Circle array
23     return circleArray;
24   }
25
26   /** Print an array of circles and their total area */
27   public static void printCircleArray(Circle3[] circleArray) {
28     System.out.printf("%-30s%-15s\n", "Radius", "Area");
29     for (int i = 0; i < circleArray.length; i++) {
30       System.out.printf("%-30f%-15f\n", circleArray[i].getRadius(),
31         circleArray[i].getArea());
32     }
33
34     System.out.println("-----------------------------------------");
35
36     // Compute and display the result
37     System.out.printf("%-30s%-15f\n", "The total area of circles is",
38       sum(circleArray));
39   }
40
41   /** Add circle areas */
42   public static double sum(Circle3[] circleArray) {
43     // Initialize sum
44     double sum = 0;
45
46     // Add areas to sum
47     for (int i = 0; i < circleArray.length; i++)
48       sum += circleArray[i].getArea();
49
50     return sum;
51   }
52 }
```

```
 Radius                                  Area
70.577708                          15648.941866
44.152266                           6124.291736
24.867853                           1942.792644
 5.680718                            101.380949
36.734246                           4239.280350
-------------------------------------------------------
The total area of circles is       28056.687544
```

程序调用createCircleArray()方法（第8行）创建一个由5个Circle对象组成的数组。本章介绍了几个Circle类。本例使用的是8.9节中介绍的Circle类。

圆的半径是使用Math.random()方法随机生成的（第19行）。createCircleArray方法返回Circle对象构成的数组（第23行）。这个数组作为参数传给printCircleArray方法，该方法显示每个圆的半径和面积，以及它们的总面积。

圆的面积之和是用sum方法计算出来的（第38行），该方法以Circle对象的数组为参数，返回的是double型的总面积值。

关键术语

accessor method(getter)（读取器方法（访问器））
action（动作）
attribute（属性）
behavior（行为）
class（类）
client（客户）
constructor（构造方法）
date field（数据域）
data-field encapsulation（数据域封装）
default constructor（默认构造方法）
dot operator（.）（圆点运算符）
instance（实例）
instance method（实例方法）
instance variable（实例变量）
instantiation（实例化）
mutator method（setter）（设置器方法（修改器））
null（空值）
no-arg constructor（无参构造方法）
object-oriented programming(OOP)（面向对象程序设计）
Unified Modeling Language(UML)（统一建模语言）
package-private(or package-access)（包私有（或包访问））
private（私有的）
property（特征）
public（公共的）
reference variable（引用变量）
reference type（引用类型）
state（状态）
static method（静态方法）
static variable（静态变量）

本章小结

- 类是对象的模板。它定义对象的属性，并提供创建对象的构造方法以及对对象进行操作的方法。
- 类也是一种数据类型。可以用它声明对象引用变量。对象引用变量中似乎存放了一个对象，但事实上，它包含的只是对该对象的引用。严格地讲，对象引用变量和对象是不同的，但是大多数情况下，它们的区别是可以忽略的。
- 对象是类的实例。可以使用new操作符创建对象，使用点运算符（.）通过对象的引用变量来访问该对象的成员。
- 实例变量或方法属于类的一个实例。它的使用与各自的实例相关联。静态变量是被同一个类的所有实例所共享的。可以在不使用实例的情况下调用静态方法。
- 类的每个实例都能访问这个类的静态变量和静态方法。然而，为清晰起见，最好使用“类名.变量”和“类名.方法”来调用静态变量和静态方法。

- 修饰符指定类、方法和数据是如何被访问的。公共的（public）类、方法或数据可以被任何客户访问，私有的（private）方法或数据只可能在类内被访问。
- 可以提供get方法或者set方法，使客户能够看到或修改数据。通俗点讲，get方法称为读取器（或访问器），set方法称为设置器（或修改器）。
- get方法具有签名public returnType getPropertyName()。如果返回类型（returnType）是boolean型，则get方法应该定义为public boolean isPropertyName()。set方法具有签名public void setPropertyName(dataType propertyValue)。
- 所有传递给方法的参数都是值传递的。对于基本类型的参数，传递的是实际值；而若参数是引用数据类型，则传递的是对象的引用。
- Java数组是一个包含基本类型值或对象类型值的对象。当创建一个对象数组时，它的元素被赋予默认值null。

复习题

8.2～8.5节

8.1 描述对象和它的定义类之间的关系。如何定义一个类？如何声明一个对象引用变量？如何创建一个对象？如何在一条语句中完成对象的声明和创建？

8.2 构造方法和普通方法之间的区别是什么？

8.3 数组是一个对象或一个基本类型值吗？数组可以包含对象类型和基本类型的元素吗？描述数组元素的默认值。

8.4 下面的程序有什么错误？

```
1 public class ShowErrors {
2   public static void main(String[] args) {
3     ShowErrors t = new ShowErrors(5);
4   }
5 }
```

a)

```
1 public class ShowErrors {
2   public static void main(String[] args) {
3     ShowErrors t = new ShowErrors();
4     t.x();
5   }
6 }
```

b)

```
1 public class ShowErrors {
2   public void method1() {
3     Circle c;
4     System.out.println("What is radius "
5       + c.getRadius());
6     c = new Circle();
7   }
8 }
```

c)

```
1 public class ShowErrors {
2   public static void main(String[] args) {
3     C c = new C(5.0);
4     System.out.println(c.value);
5   }
6 }
7
8 class C {
9   int value = 2;
10 }
```

d)

8.5 下面的代码有什么错误？

```
1 class Test {
2   public static void main(String[] args) {
3     A a = new A();
4     a.print();
5   }
6 }
7
8 class A {
9   String s;
10
11   A(String s) {
12     this.s = s;
```

```
13   }
14
15   public void print() {
16     System.out.print(s);
17   }
18 }
```

8.6 下面代码的输出是什么？

```
public class Foo {
  private boolean x;

  public static void main(String[] args) {
    Foo foo = new Foo();
    System.out.println(foo.x);
  }
}
```

8.6节

8.7 如何为当前时间创建一个Date？如何显示当前时间？

8.8 如何创建一个JFrame，如何在框架中设置一个标题以及显示这个框架？

8.9 哪些包中包含类Date、JFrame、JOptionPane、System和Math？

8.7节

8.10 假设Foo类在图a中定义，f是Foo的一个实例，那么图b中的哪些语句是正确的？

```
public class Foo {
  int i;
  static String s;

  void imethod() {
  }

  static void smethod() {
  }
}
```

a)

```
System.out.println(f.i);
System.out.println(f.s);
f.imethod();
f.smethod();
System.out.println(Foo.i);
System.out.println(Foo.s);
Foo.imethod();
Foo.smethod();
```

b)

8.11 如果合适的话，在出现?的位置添加static关键字。

```
public class Test {
  private int count;

  public ? void main(String[] args) {
    ...
  }

  public ? int getCount() {
    return count;
  }

  public ? int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
      result *= i;

    return result;
  }
}
```

8.12 能否从静态方法中调用实例方法或引用一个实例变量？能否从实例方法中调用静态方法或引用一个静态变量？下面代码的错误在哪里？

```
1 public class Foo {
2   public static void main(String[] args) {
3     method1();
4   }
```

```
5
6   public void method1() {
7     method2();
8   }
9
10  public static void method2() {
11    System.out.println("What is radius " + c.getRadius());
12  }
13
14  Circle c = new Circle();
15 }
```

8.8～8.9节

8.13 什么是访问器方法？什么是修改器方法？访问器方法和修改器方法的命名习惯是什么？

8.14 数据域封装的优点是什么？

8.15 在下面的代码中，Circle类中的radius是私有的，而myCircle是Circle类的一个对象，下面强调的代码会导致什么问题吗？解释为什么。

```
public class Circle {
  private double radius = 1.0;

  /** Find the area of this circle */
  public double getArea() {
    return radius * radius * Math.PI;
  }

  public static void main(String[] args) {
    Circle myCircle = new Circle();
    System.out.println("Radius is " + myCircle.radius);
  }
}
```

8.10节

8.16 描述传递基本类型参数和传递引用类型参数的区别。给出下面程序的输出：

```
public class Test {
  public static void main(String[] args) {
    Count myCount = new Count();
    int times = 0;

    for (int i = 0; i < 100; i++)
      increment(myCount, times);

    System.out.println("count is " + myCount.count);
    System.out.println("times is " + times);
  }

  public static void increment(Count c, int times) {
    c.count++;
    times++;
  }
}
```

```
public class Count {
  public int count;

  public Count(int c) {
    count = c;
  }

  public Count() {
    count = 1;
  }
}
```

8.17 显示下面程序的输出：

```
public class Test {
  public static void main(String[] args) {
    Circle circle1 = new Circle(1);
    Circle circle2 = new Circle(2);

    swap1(circle1, circle2);
    System.out.println("After swap1: circle1 = " +
      circle1.radius + " circle2 = " + circle2.radius);

    swap2(circle1, circle2);
    System.out.println("After swap2: circle1 = " +
```

```
        circle1.radius + " circle2 = " + circle2.radius);
    }
    public static void swap1(Circle x, Circle y) {
      Circle temp = x;
      x = y;
      y = temp;
    }

    public static void swap2(Circle x, Circle y) {
      double temp = x.radius;
      x.radius = y.radius;
      y.radius = temp;
    }
  }

  class Circle {
    double radius;

    Circle(double newRadius) {
      radius = newRadius;
    }
  }
```

8.18 显示下面代码的输出：

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a[0], a[1]);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }
}
```

a)

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int[] a) {
    int temp = a[0];
    a[0] = a[1];
    a[1] = temp;
  }
}
```

b)

```
public class Test {
  public static void main(String[] args) {
    T t = new T();
    swap(t);
    System.out.println("e1 = " + t.e1
      + " e2 = " + t.e2);
  }

  public static void swap(T t) {
    int temp = t.e1;
    t.e1 = t.e2;
    t.e2 = temp;
  }
}

class T {
  int e1 = 1;
  int e2 = 2;
}
```

c)

```
public class Test {
  public static void main(String[] args) {
    T t1 = new T();
    T t2 = new T();
    System.out.println("t1's i = " +
      t1.i + " and j = " + t1.j);
    System.out.println("t2's i = " +
      t2.i + " and j = " + t2.j);
  }
}

class T {
  static int i = 0;
  int j = 0;

  T() {
    i++;
    j = 1;
  }
}
```

d)

8.19 下面程序的输出是什么？

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = null;
    m1(date);
    System.out.println(date);
  }

  public static void m1(Date date) {
    date = new Date();
  }
}
```

a)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = new Date(7654321);
  }
}
```

b)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date.setTime(7654321);
  }
}
```

c)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = null;
  }
}
```

d)

8.11节

8.20 下面代码的错误在哪里？

```
1 public class Test {
2   public static void main(String[] args) {
3     java.util.Date[] dates = new java.util.Date[10];
4     System.out.println(dates[0]);
5     System.out.println(dates[0].toString());
6   }
7 }
```

编程练习题

教学注意 第8～14章的练习要达到下面三个目标：

- 设计类并画出UML类图。
- 实现UML中的类。
- 使用类开发应用程序。

8.2～8.5节

8.1 （矩形类Rectangle）遵从8.2节中Circle类的例子，设计一个名为Rectangle的类表示矩形。这个类包括：

- 两个名为width和height的double型数据域，它们分别表示矩形的宽和高。width 和height的默认值都为1。
- 创建默认矩形的无参构造方法。
- 一个创建width和height为指定值的矩形的构造方法。
- 一个名为getArea()的方法返回这个矩形的面积。

- 一个名为getPerimeter()的方法返回周长。

画出该类的UML图。实现这个类。编写一个测试程序，创建两个Rectangle对象——一个矩形的宽为4而高为40，另一个矩形的宽为3.5而高为35.9。依照每个矩形的宽、高、面积和周长的顺序显示。

8.2 （股票类Stock）遵循8.2节中Circle类的例子，设计一个名为Stock的类。这个类包括：

- 一个名为symbol的字符串数据域表示股票代码。
- 一个名为name的字符串数据域表示股票名字。
- 一个名为previousClosingPrice的double型数据域，它存储的是前一日的股票值。
- 一个名为currentPrice的double型数据域，它存储的是当时的股票值。
- 创建一支有特定代码和名字的股票的构造方法。
- 一个名为getChangePercent()的方法返回从previousClosingPrice变化到currentPrice的百分比。

画出该类的UML图。实现这个类。编写一个测试程序，创建一个Stock对象，它的股票代码是Java，股票名字为Sun Microsystems Inc，前一日收盘价是4.5。设置新的当前值为4.35，然后显示市值变化的百分比。

8.6节

*8.3 （使用日期类Date）编写程序创建一个Date对象，设置它的流逝时间分别为10000、100000、1000000、10000000、100000000、1000000000、10000000000、100000000000，然后使用toString()方法显示日期。

*8.4 （使用随机类Random）编写一个程序，创建种子是1000的Random对象，然后使用nextInt(100)方法显示0到100之间前50个随机整数。

*8.5 （使用公历类GregorianCalendar）Java API有一个在包java.util中的类GregorianCalendar，可以使用它获得某个日期的年、月、日。它的无参构造方法构建一个当前日期的实例，get(GregorianCalendar.YEAR)、get(GregorianCalendar.MONTH)和get(GregorianCalendar.DAY_OF_MONTH)方法返回年、月和日。编写一个程序完成两个任务：

- 显示当前的年、月和日。
- GregorianCalendar类有方法setTimeInMillis(long)，可以用它来设置从1970年1月1日算起的一个特定时间。将这个值设置为1234567898765L，然后显示这个年、月和日。

8.7～8.9节

**8.6 （显示日历）改写程序清单5-12中的PrintCalendar类，在消息对话框中显示日历。因为输出是由该类中的几个静态方法产生的，所以可以定义一个静态的String型变量output，用以存储输出结果，并且在消息对话框中显示它。

8.7 （账户类Account）设计一个名为Account的类，它包括：

- 一个名为id的int类型私有账户数据域（默认值为0）。
- 一个名为balance的double类型私有账户数据域（默认值为0）。
- 一个名为annualInterestRate的double类型私有数据域存储当前利率（默认值为0）。假设所有的账户都有相同的利率。
- 一个名为dateCreated的Date类型私有数据域存储账户的开户日期。
- 一个能创建默认账户的无参构造方法。
- 一个能创建带特定id和初始余额的账户的构造方法。
- id、balance和annualInterstRate的访问器和修改器。
- dateCreated的访问器。
- 一个名为getMonthlyInterestRate()的方法返回月利率。

- 一个名为withDraw的方法从账户提取特定数额。
- 一个名为deposit的方法向账户存储特定数额。

画出该类的UML图。实现这个类。编写一个测试程序，创建一个账户ID为1122、余额为20 000美元、年利率为4.5%的Account对象。使用withdraw方法取款2500美元，使用deposit方法存款3000美元，然后打印余额、月利息以及这个账户的开户日期。

8.8 （风扇类Fan）设计一个名为Fan的类来表示一个风扇。这个类包括：

- 三个名为SLOW、MEDIUM和FAST而值是1、2和3的常量表示风扇的速度。
- 一个名为speed的int类型私有数据域表示风扇的速度（默认值为SLOW）。
- 一个名为on的boolean类型私有数据域表示风扇是否打开（默认值为false）。
- 一个名为radius的double类型私有数据域表示风扇的半径（默认值为5）。
- 一个名为color的string类型数据域表示风扇的颜色（默认值为blue）。
- 这四个数据域的访问器和修改器。
- 一个创建默认风扇的无参构造方法。
- 一个名为toString()的方法返回描述风扇的字符串。如果风扇是打开的，那么该方法在一个组合的字符串中返回风扇的速度、颜色和半径。如果风扇没有打开，该方法就会返回一个由“fan is off”和风扇颜色及半径组合成的字符串。

画出该类的UML图。实现这个类。编写一个测试程序，创建两个Fan对象。将第一个对象设置为最大速度、半径为10、颜色为yellow、状态为打开。将第二个对象设置为中等速度、半径为5、颜色为blue、状态为关闭。通过调用它们的toString方法显示这些对象。

**8.9 （几何方面：正n边形）在一个正n边形中，所有边的长度都相同，且所有角的度数都相同（即这个多边形是等边等角的）。设计一个名为RegularPolygon的类，该类包括：

- 一个名为n的int型私有数据域定义多边形的边数，默认值为3。
- 一个名为side的double型私有数据域存储边的长度，默认值为1。
- 一个名为x的double型私有数据域，它定义多边形中点的x坐标，默认值为0。
- 一个名为y的double型私有数据域，它定义多边形中点的y坐标，默认值为0。
- 一个创建带默认值的正多边形的无参构造方法。
- 一个能创建带指定边数和边长度、中心在（0,0）的正多边形的构造方法。
- 一个能创建带指定边数和边长度、中心在（x,y）的正多边形的构造方法。
- 所有数据域的访问器和修改器。
- 一个返回多边形周长的方法getPerimeter()。
- 一个返回多边形面积的方法getArea()。计算正多边形面积的公式是：

$$\text{面积} = \frac{n \times s^2}{4 \times \tan\left(\frac{p}{n}\right)}$$

画出该类的UML图。实现这个类。编写一个测试程序，分别使用无参构造方法、RegularPolygon(6,4)和RegularPolygon(10,4,5.6,7.8)创建三个RegularPolygon对象。显示每个对象的周长和面积。

*8.10 （代数方面：二次方程式）为二次方程式$ax^2+bx+c=0$设计一个名为QuadraticEquation的类。这个类包括：

- 代表三个系数的私有数据域a、b和c。
- 一个参数为a、b和c的构造方法。
- a、b、c的三个get方法。
- 一个名为getDiscriminant()的方法返回判别式，b^2-4ac。
- 一个名为getRoot1()和getRoot2()的方法返回等式的两个根：

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ and } r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

这些方法只有在判别式为非负数时才有用。如果判别式为负，这些方法返回0。

画出该类的UML图。实现这个类。编写一个测试程序，提示用户输入*a*、*b*和*c*的值，然后显示判别式的结果。如果判别式为正数，显示两个根；如果判别式为0，显示一个根；否则，显示“The equation has no roots.”（这个方程无根）。参见练习题3.1的运行示例。

*8.11 （代数方面：2×2的线性方程）为一个2×2的线性方程设计一个名为LinearEquation的类：

$$ax + by = e \qquad cx + dy = f \qquad x = \frac{ed - bf}{ad - bc} \qquad y = \frac{af - ec}{ad - bc}$$

这个类包括：

- 私有数据域a、b、c、d、e和f。
- 一个参数为a、b、c、d、e、f的构造方法。
- a、b、c、d、e、f的六个get方法。
- 一个名为isSolvable()的方法，如果*ad*−*bc*不为0则返回true。
- 方法getX()和getY()返回这个方程的解。

画出该类的UML图。实现这个类。编写一个测试程序，提示用户输入a、b、c、d、e、f的值，然后显示它的结果。如果*ad*−*bc*为0，就报告“The equation has no solution.”。参见练习题3.3的运行示例。

**8.12 （几何方面：交点）假设两个线段相交。第一个线段的两个端点是（x1,y1）和（x2,y2），第二个线段的两个端点是（x3,y3）和（x4,y4）。编写一个程序，提示用户输入这四个端点，然后显示它们的交点。

提示 使用前面练习题中的LinearEquation类。

```
Enter the endpoints of the first line segment: 2.0 2.0 0 0 Enter
Enter the endpoints of the second line segment: 0 2.0 2.0 0 Enter
The intersecting point is: (1.0, 1.0)
```

**8.13 （位置类Location）设计一个名为Location的类，定位二维数组中的最大值及其位置。这个类包括公共的数据域row、column和maxValue，二维数组中的最大值及其下标用int型的row和column以及double型的maxValue存储。

编写下面的方法，返回一个二维数组中最大值的位置。

```
public static Location locateLargest(double[][] a)
```

返回值是一个Location的实例。编写一个测试程序，提示用户输入一个二维数组，然后显示这个数组中的最大元素。下面是一个运行示例：

```
Enter the number of rows and columns of the array: 3 4 Enter
Enter the array:
23.5 35 2 10 Enter
4.5 3 45 3.5 Enter
35 44 5.5 9.6 Enter
The location of the largest element is 45 at (1, 2)
```