第10章
Introduction to Java Programming, 8E

关于对象的思考

学习目标

- 从不可变类创建不可变对象以保护对象的内容（10.2节）。
- 判断类中出现的变量的作用域（10.3节）。
- 使用关键字this指代调用对象自己（10.4节）。
- 应用类的抽象来开发软件（10.5节）。
- 探索面向过程范式和面向对象范式的不同之处（10.6节）。
- 开发用于建模对象之间组合关系的类（10.7节）。
- 使用面向对象范式设计程序（10.8～10.10节）。
- 按照类的设计原则来设计类（10.11节）。

10.1 引言

前面两章已经介绍了对象和类。你也学习了如何定义类、创建对象以及使用Java API中的一些类的对象（例如：Date、Random、String、StringBuilder、File、Scanner、PrintWriter）。本书的方法是在教授面向对象程序设计之前，先讲述解决问题的技术和基本程序设计的技术。本章将给出面向过程和面向对象程序设计的不同之处。你将会看到面向对象程序设计的优点，并学习如何高效地使用它。

这里，我们的焦点放在类的设计上。我们将使用几个例子来诠释面向对象方法的优点。这些例子包括如何在应用程序中设计新类以及如何使用这些类。下面首先介绍一些支持这些例子的语言特征。

10.2 不可变对象和类

通常，创建一个对象后，它的内容是允许随后改变的。有时候，也需要创建一个一旦创建，其内容就不能再改变的对象。我们称这种对象为一个不可变对象（immutable object），而它的类就称为不可变类（immutable class）。例如：String类就是不可变的。如果把程序清单8-9中Circle类的set方法删掉，该类就变成不可变类，因为半径是私有的，所以没有set方法，它的值就不能再改变。

如果一个类是不可变的，那么它的所有数据域必须都是私有的，而且没有对任何一个数据域提供公共的set方法。一个类的所有数据都是私有的且没有修改器，并不意味着它一定是不可变类。例如：下面的Student类，它的所有数据域都是私有的，而且也没有set方法，但它不是一个不可变的类。

```
1  public class Student {
2    private int id;
3    private String name;
4    private java.util.Date dateCreated;
5
6    public Student(int ssn, String newName) {
7      id = ssn;
8      name = newName;
9      dateCreated = new java.util.Date();
10   }
11
12   public int getId() {
13     return id;
14   }
15
```

```
16   public String getName() {
17     return name;
18   }
19
20   public java.util.Date getDateCreated() {
21     return dateCreated;
22   }
23 }
```

如下面的代码所示，使用getDateCreated()方法返回数据域dateCreated。它是对Date对象的一个引用。通过这个引用，可以改变dateCreated的值。

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, "John");
    java.util.Date dateCreated = student.getDateCreated();
    dateCreated.setTime(200000); // Now dateCreated field is changed!
  }
}
```

要使一个类成为不可变的，它必须满足下面的需求：

- 所有数据域都是私有的。
- 没有修改器方法。
- 没有一个访问器方法，它会返回一个指向可变数据域的引用。

10.3 变量的作用域

在第5章中讨论了局部变量和它们的作用域。局部变量的声明和使用都在一个方法的内部。本节将在类的范围内讨论所有变量的作用域规则。

一个类的实例变量和静态变量称为*类变量*（class's variables）或*数据域*（data field）。在方法内部定义的变量称为局部变量。不管在何处声明，类变量的作用域都是整个类。类的变量和方法可以在类中以任意顺序出现，如图10-1a所示。当一个数据域是基于对另一个数据域的引用来进行初始化的情况例外。在这种情况下，必须首先声明另一个数据域，如图10-1b所示。为保持一致性，本书在类的开头就声明数据域。

```
public class Circle {
  public double findArea() {
    return radius * radius * Math.PI;
  }

  private double radius = 1;
}
```

a）可以按任意顺序声明变量radius和方法findArea()

```
public class Foo {
  private int i;
  private int j = i + 1;
}
```

b）i必须在j之前声明，因为j的初始值依赖于i

图10-1 类的成员可以按任意顺序声明，只有一种例外情况

类变量只能声明一次，但是在一个方法内不同的非嵌套块中，可以多次声明相同的变量名。

如果一个局部变量和一个类变量具有相同的名字，那么局部变量优先，而同名的类变量将被*隐藏*（hidden）。例如：在下面的程序中，x被定义为一个实例变量，也在方法中被定义为局部变量。

```
public class Foo {
  private int x = 0; // Instance variable
  private int y = 0;

  public Foo() {
  }

  public void p() {
    int x = 1; // Local variable
```

```
    System.out.println("x = " + x);
    System.out.println("y = " + y);
  }
}
```

假设f是Foo的一个实例，那么f.p()的打印输出是什么呢？f.p()的打印输出是：x为1，y为0。其原因如下：

- x被声明为类中初值为0的数据域，但是它在方法p()中又被声明了一次，初值为1。System.out.println语句中引用的x是后者。
- y在方法p()的外部声明，但在内部也是可访问的。

提示 为避免混淆和错误，除了方法中的参数，不要将实例变量或静态变量的名字作为局部变量名。

10.4 this引用

关键字this是指向调用对象本身的引用名。一种常见的用法就是引用类的隐藏数据域（hidden data field）。例如：在数据域的set方法中，经常将数据域名用作参数名。在这种情况下，这个数据域在set方法中被隐藏。为了给它设置新值，需要在方法中引用隐藏的数据域名。隐藏的静态变量可以简单地通过“类名.静态变量”的方式引用。隐藏的实例变量就需要使用关键字this来引用，如图10-2a所示。

```
public class Foo {
  int i = 5;
  static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    Foo.k = k;
  }
}
```

a)

```
Suppose that f1 and f2 are two objects of Foo.

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2
```

b)

图10-2 关键字this是指调用方法的对象

关键字this给出一种指代对象的方法，这样，可以在实例方法代码中调用实例方法。this.i=i这一行的意思是“将参数i的值赋给调用对象的数据域i”。关键字this是指调用实例方法setI的对象，如图10-2b所示。Foo.k=k这一行的意思是将参数k的值赋给这个类的静态数据域k，k是被类的所有对象所共享的。

关键字this的另一个常用方法是让构造方法调用同一个类的另一个构造方法。例如，可以如下改写Circle类：

```
public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }                          ← 显式使用this来引用被创建对象的数据域radius

  public Circle() {
    this(1.0);
  }                          ← 使用this调用另一个构造方法

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }
}
```

每个实例变量都属于一个this表示的实例，通常这个this是被忽略的

在第二个构造方法中，this(1.0)这一行调用带double值参数的第一个构造方法。

提示 如果一个类有多个构造方法，最好尽可能使用this（参数列表）实现它们。通常，无参数或参数少的构造方法可以用this（参数表）调用参数多的构造方法。这样做通常可以简化代码，使类易于阅读和维护。

注意 Java要求在构造方法中，语句this（参数列表）应在任何其他语句之前出现。

10.5 类的抽象和封装

在第5章中，已经学习了方法的抽象，以及如何在程序开发中使用它。Java提供了多层次的抽象。*类抽象*（class abstraction）是将类的实现和使用分离。类的创建者提供类的描述，让使用者明白如何才能使用类。从类外可以访问的全部方法和数据域，以及期望这些成员如何行动的描述，合称为*类的合约*（class's contract）。如图10-3所示，类的使用者不需要知道类是如何实现的。实现的细节经过封装，对用户隐藏起来，这称为*类的封装*（class encapsulation）。例如：可以创建一个Circle对象，并且可以在不知道面积是如何计算出来的情况下，求出这个圆的面积。

图10-3 类抽象将类的实现与类的使用分离

类的抽象和封装是一个问题的两个方面。现实生活中的许多例子都可以说明类抽象的概念。例如：考虑建立一个计算机系统。个人计算机有很多组件——CPU、内存、磁盘、主板和风扇等。每个组件都可以看做是一个有属性和方法的对象。要使各个组件一起工作，只需要知道每个部件是怎么用的，以及是如何与其他部件进行交互的。无须了解这些组件内部是如何工作的。内部功能的实现被封装起来，对你是隐藏的。所以，你可以组装一台计算机，而不需要了解每个组件是如何实现功能的。

对计算机系统的模拟准确地反映了面向对象方法。每个部件可以看成组件类的对象。例如：你可能已经建立了一个类，模拟用在计算机上的各种类型的风扇，它具有风扇尺寸和速度等属性，还有像开始和停止这样的方法。一个具体的风扇就是该类具有特定属性值的实例。

将得到一笔贷款作为另一个例子。一笔具体的贷款可以看做贷款类Loan的一个对象，利率、贷款额以及还贷时间都是它的数据属性，计算每月偿还金额和总偿还金额是它的方法。当你购买一辆汽车时，就用贷款利率、贷款额和还贷时间实例化这个类，创建一个贷款对象。然后，就可以使用这些方法计算贷款的月偿还额和总偿还额。作为一个贷款类Loan的用户，是不需要知道这些方法是如何实现的。

程序清单2-8给出计算贷款偿还额的程序。这个程序不能在其他程序中重用。解决这个问题的一种方式就是定义计算月偿还额和总偿还额的静态方法。但是，这个解决方案是有限的。假设希望将一个数据和这个贷款联系起来，最理想的方法是创建一个对象，将这个数据和关于贷款信息的属性绑定在一起。图10-4给出Loan类的UML类图。

在图10-4中的UML图被当做是Loan类的合约。贯穿本书，你将扮演两个角色，一个是类的用户，一个是类的开发者。用户可以在不知道类是如何实现的情况下使用类。假设Loan类是可用的。我们从编写一个使用Loan类的测试类开始。

程序清单10-1 TestLoanClass.java

```
1 import java.util.Scanner;
2
3 public class TestLoanClass {
4   /** Main method */
5   public static void main(String[] args) {
6     // Create a Scanner
```

```
    Scanner input = new Scanner(System.in);

    // Enter yearly interest rate
    System.out.print(
      "Enter yearly interest rate, for example, 8.25: ");
    double annualInterestRate = input.nextDouble();

    // Enter number of years
    System.out.print("Enter number of years as an integer: ");
    int numberOfYears = input.nextInt();

    // Enter loan amount
    System.out.print("Enter loan amount, for example, 120000.95: ");
    double loanAmount =  input.nextDouble();

    // Create a Loan object
    Loan loan =
      new Loan(annualInterestRate, numberOfYears, loanAmount);

    // Display loan date, monthly payment, and total payment
    System.out.printf("The loan was created on %s\n" +
      "The monthly payment is %.2f\nThe total payment is %.2f\n",
      loan.getLoanDate().toString(), loan.getMonthlyPayment(),
      loan.getTotalPayment());
  }
}
```

```
Enter yearly interest rate, for example, 8.25: 2.5 Enter
Enter number of years as an integer: 5 Enter
Enter loan amount, for example, 120000.95: 1000 Enter
The loan was created on Sat Jun 10 21:12:50 EDT 2006
The monthly payment is 17.74
The total payment is 1064.84
```

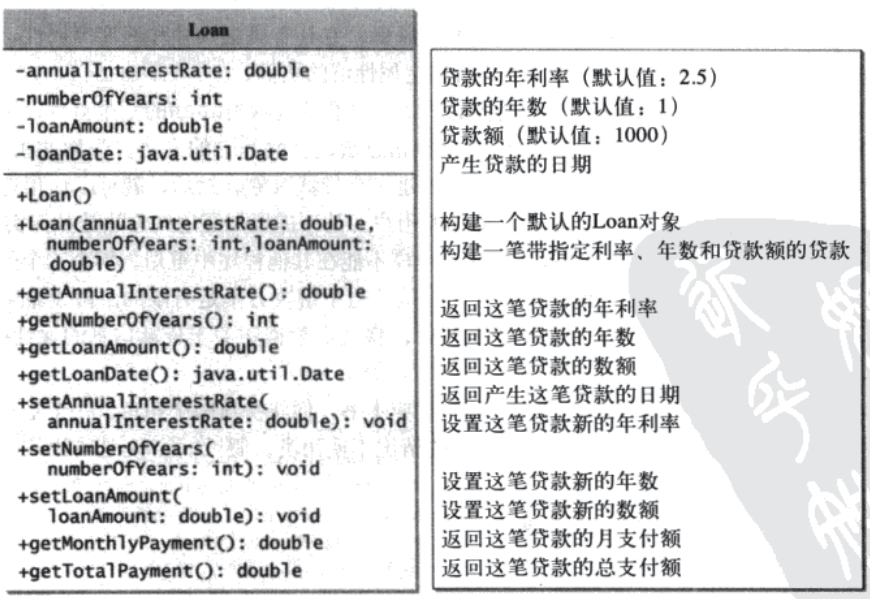


图10-4 Loan类对贷款的属性和行为建模

main方法读取利率、还贷时间（以年为单位）、贷款总额；创建一个Loan对象；然后使用Loan类中的实例方法获取月偿还额（第29行）和总偿还额（第30行）。

Loan类可以在程序清单10-2中实现。

程序清单10-2 Loan.java

```
1  public class Loan {
2    private double annualInterestRate;
3    private int numberOfYears;
4    private double loanAmount;
5    private java.util.Date loanDate;
6
7    /** Default constructor */
8    public Loan() {
9      this(2.5, 1, 1000);
10   }
11
12   /** Construct a loan with specified annual interest rate,
13       number of years, and loan amount
14     */
15   public Loan(double annualInterestRate, int numberOfYears,
16       double loanAmount) {
17     this.annualInterestRate = annualInterestRate;
18     this.numberOfYears = numberOfYears;
19     this.loanAmount = loanAmount;
20     loanDate = new java.util.Date();
21   }
22
23   /** Return annualInterestRate */
24   public double getAnnualInterestRate() {
25     return annualInterestRate;
26   }
27
28   /** Set a new annualInterestRate */
29   public void setAnnualInterestRate(double annualInterestRate) {
30     this.annualInterestRate = annualInterestRate;
31   }
32
33   /** Return numberOfYears */
34   public int getNumberOfYears() {
35     return numberOfYears;
36   }
37
38   /** Set a new numberOfYears */
39   public void setNumberOfYears(int numberOfYears) {
40     this.numberOfYears = numberOfYears;
41   }
42
43   /** Return loanAmount */
44   public double getLoanAmount() {
45     return loanAmount;
46   }
47
48   /** Set a new loanAmount */
49   public void setLoanAmount(double loanAmount) {
50     this.loanAmount = loanAmount;
51   }
52
53   /** Find monthly payment */
54   public double getMonthlyPayment() {
55     double monthlyInterestRate = annualInterestRate / 1200;
56     double monthlyPayment = loanAmount * monthlyInterestRate / (1 -
57       (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
58     return monthlyPayment;
```

```
59   }
60
61   /** Find total payment */
62   public double getTotalPayment() {
63     double totalPayment = getMonthlyPayment() * numberOfYears * 12;
64     return totalPayment;
65   }
66
67   /** Return loan date */
68   public java.util.Date getLoanDate() {
69     return loanDate;
70   }
71 }
```

从类的开发者的角度来看，设计类是为了让很多不同的用户所使用。为了在更大的应用范围内使用类，类应该通过构造方法、属性和方法提供不同方式的定制。

Loan类包含两个构造方法、四个get方法、三个set方法，以及求月偿还额和总偿还额的方法。可以通过使用无参构造方法或者带三个参数：年利率、年数和贷款额的构造方法来构造一个Loan对象。当创建一个贷款对象时，它的数据存储在loanDate域中。getLoanDate方法返回日期。三个get方法：getAnnualInterest、getNumberOfYears和getLoanAmount分别返回年利率、还款时间以及贷款总额。这个类的所有数据属性和方法都被绑定到Loan类的某个特定实例。因此，它们都是实例变量或者方法。

重要教学提示 Loan类的UML图如图10-4所示。学生应该从编写使用Loan类的测试程序开始，即使它们不知道这个Loan类是如何执行的。它有三个优点：

1）它演示开发类和使用类是两个不同的任务。

2）它能使你跳过某个类的复杂实现，而不干扰整本书的顺序。

3）如果你通过使用它熟悉了类，那么你更容易学会如何实现一个类。

从现在开始的所有例子，你都可以先从这个类创建一个对象，然后尝试使用它的方法，之后就将注意力放在它的实现上。

10.6 面向对象的思考

第1～7章介绍使用循环、方法和数组来解决问题的基本程序设计技术。这些技术的学习为面向对象程序设计打下坚实的基础。类为构建可重用软件提供了更高的灵活性和更多的模块化。本节使用面向对象方法来改进第3章中介绍的一个问题的解决方案。在这个改进的过程中，可以看到面向过程程序设计和面向对象程序设计的不同，也可以看出使用对象和类来开发可重用代码的优势。

程序清单3-5给出计算身体质量指数的程序。它的代码不能在其他程序中重用。为使之具备可重用性，定义一个静态方法计算身体质量指数，如下所示：

```
public static double getBMI(double weight, double height)
```

这个方法对于计算给定体重和身高的身体质量指数是很有用的。但是，它是有局限性的。假设需要将体重和身高同一个人的名字与出生日期相关联。可以分别声明几个变量来存储这些值。但是这些值不是紧密耦合在一起的。将它们耦合在一起的理想方法就是创建一个包含它们的对象。因为这些值都被绑定到单独的对象上，所以它们应该存储在实例数据域中。可以定义一个名为BMI的类，如图10-5所示。

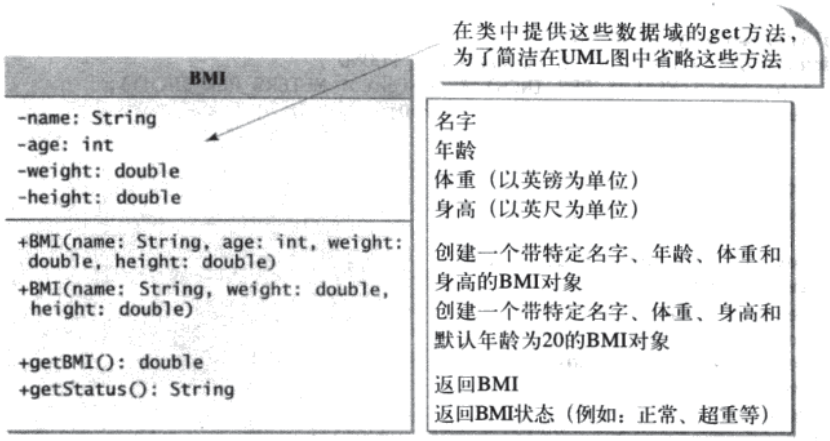


图10-5 BMI类封装BMI信息

假设BMI类是可用的。程序清单10-3给出使用这个类的测试程序。

程序清单10-3 UseBMIClass.java

```
1 public class UseBMIClass {
2   public static void main(String[] args) {
3     BMI bmi1 = new BMI("John Doe", 18, 145, 70);
4     System.out.println("The BMI for " + bmi1.getName() + " is "
5       + bmi1.getBMI() + " " + bmi1.getStatus());
6
7     BMI bmi2 = new BMI("Peter King", 215, 70);
8     System.out.println("The BMI for " + bmi2.getName() + " is "
9       + bmi2.getBMI() + " " + bmi2.getStatus());
10  }
11 }
```

```
The BMI for John Doe is 20.81 normal weight
The BMI for Peter King is 30.85 seriously overweight
```

第3行为John Doe创建一个对象bmi1，而第7行为Peter King创建一个对象bmi2。可以使用实例方法getName()、getBMI()和getStatus()返回一个BMI对象中的BMI信息。

BMI类可以在程序清单10-4中实现。

程序清单10-4 BMI.java

```
1 public class BMI {
2   private String name;
3   private int age;
4   private double weight; // in pounds
5   private double height; // in inches
6   public static final double KILOGRAMS_PER_POUND = 0.45359237;
7   public static final double METERS_PER_INCH = 0.0254;
8
9   public BMI(String name, int age, double weight, double height) {
10    this.name = name;
11    this.age = age;
12    this.weight = weight;
13    this.height = height;
14  }
15
16  public BMI(String name, double weight, double height) {
17    this(name, 20, weight, height);
18  }
```

```
19
20   public double getBMI() {
21     double bmi = weight * KILOGRAMS_PER_POUND /
22       ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
23     return Math.round(bmi * 100) / 100.0;
24   }
25
26   public String getStatus() {
27     double bmi = getBMI();
28     if (bmi < 16)
29       return "seriously underweight";
30     else if (bmi < 18)
31       return "underweight";
32     else if (bmi < 24)
33       return "normal weight";
34     else if (bmi < 29)
35       return "overweight";
36     else if (bmi < 35)
37       return "seriously overweight";
38     else
39       return "gravely overweight";
40   }
41
42   public String getName() {
43     return name;
44   }
45
46   public int getAge() {
47     return age;
48   }
49
50   public double getWeight() {
51     return weight;
52   }
53
54   public double getHeight() {
55     return height;
56   }
57 }
```

使用体重和身高来计算BMI的数学公式在3.10节中给出了。实例方法`getBMI()`返回BMI。因为体重和身高是对象的实例数据域，`getBMI()`方法可以使用这些属性来计算对象的BMI。

实例方法`getStatus()`返回解释BMI的字符串。这个解释也在3.10节中给出了。

这个例子演示了面向对象范式比面向过程范式有优势的地方。面向过程范式重在设计方法。面向对象范式将数据和方法都组合在对象中。使用面向对象范式的软件设计重在对象和对象中上的操作。面向对象方法结合了面向过程范式的功能以及将数据和操作集成在对象中的特性。

在面向过程程序设计中，数据和数据上的操作是分离的，而且这里的方法论要求给方法发送数据。面向对象程序设计将数据和对它们的操作都放在一个对象中。这个方法解决了很多面向过程程序设计中的问题。面向对象程序设计方法按照镜像到真实世界的方式组织程序，在真实世界中，所有的对象都是和属性及动作相关联的。使用对象提高了软件的可重用性，并且使程序更易于开发和维护。Java程序设计涉及的是对对象的思考，一个Java程序可以看做是一个相互操作的对象集合。

10.7 对象的组合

一个对象可以包含另一个对象。这两个对象之间的关系称为**组合**（composition）。在程序清单10-4中，定义的`BMI`类包含了一个`String`数据域。`BMI`和`String`之间的关系就是组合。

组合实际上是聚集关系的一种特殊形式。聚集模拟了具有（has-a）关系，表示两个对象之间的归属

关系。归属关系中的所有者对象称为聚集对象（aggregating object），而它的类称为聚集类（aggregating class）。归属关系中的从属对象称为被聚集对象（aggregated object），而它的类称为被聚集类（aggregated class）。

一个对象可以被几个其他聚集对象所拥有。如果一个对象只归属于一个聚集对象，那么它和聚集对象之间的关系就称为组合（composition）。例如："一个学生有一个名字"就是学生类Student与名字类Name之间的一个组合关系，而"一个学生有一个地址"是学生类Student与地址类Address之间的一个聚集关系，因为一个地址可以被几个学生所共享。在UML中，附加在聚集类（例如：Student）上的实心菱形表示它和被聚集类（例如：Name）之间具有组合关系；而附加在聚集类（例如：Student）上的空心菱形表示它与被聚集类（例如：Address）之间具有聚集关系，如图10-6所示。

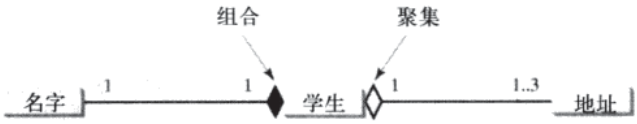


图10-6 一个学生有一个名字和一个地址

在一个关系中的每个类可以指定一个多重性（multiplicity）。这个多重性可能是一个数字或者一个数字范围，它表明关系中要涉及的类的对象个数。字符*意味着无限多个对象，而数字范围m..n意味着对象的个数应该在m和n之间，包括m和n。在图10-6中，每个学生只能有一个地址，而每个地址最多可以被3个学生共享。每个学生都有一个名字，而每个学生的名字都是唯一的。

聚集关系通常被表示为聚集类中的一个数据域。例如：图10-6中的关系可以如下表示：

```
public class Name {
  ...
}
```

被聚集类

```
public class Student {
  private Name name;
  private Address address;

  ...
}
```

聚集类

```
public class Address {
  ...
}
```

被聚集类

聚集可以存在于同一类的多个对象之间。例如：一个人可能有一个管理者，如图10-7所示。

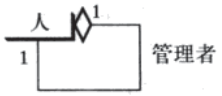


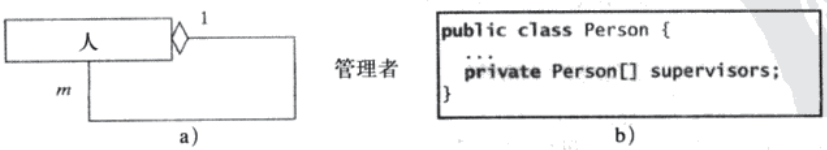
图10-7 一个人可能有一个管理者

在图10-7所示的关系"一个人可能有一个管理者"中，管理者可以如下表示为Person类的一个数据域：

```
public class Person {
  // The type for the data is the class itself
  private Person supervisor;

  ...
}
```

如果一个人有几个管理者，如图10-8a所示，可以用一个数组存储管理者，如图10-8b所示。



a) b)

图10-8 一个人可能会有几个管理者

注意 由于聚集和组合关系都以相似方式用类来表示，许多教材都不区分它们，将两者都称为组合。

10.8 设计类Course

本书的宗旨是“通过例子来教学，通过动手来学习”(teaching by example and learning by doing)。本书提供了多种类的例子来演示面向对象程序设计。10.8～10.10节将给出设计类的附加例子。

假设需要处理课程信息。每门课程都有一个名字以及选课的学生，要能够向/从这个课程添加/删除一个学生。可以使用一个类来对课程建模，如图10-9所示。

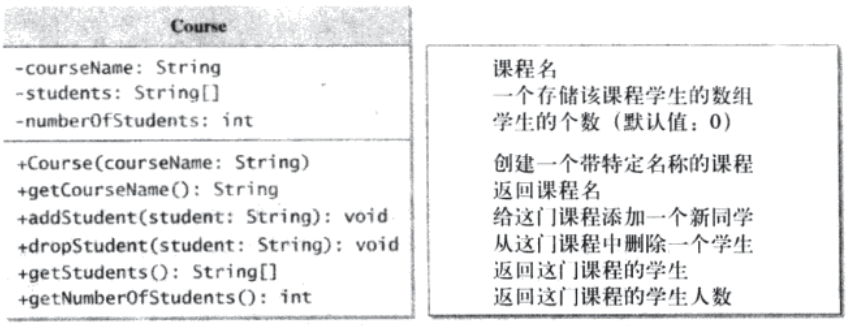


图10-9 Course类对课程建模

向构造方法Course(String name)传递一门课程的名称来创建一个Course对象。你可以使用addStudent(String student)方法向某门课程添加学生，使用dropStudent(String student)方法从某门课程中删掉一个学生，而使用getStudents()方法可以返回选这门课程的所有学生。假设该类是可用的。程序清单10-5就给出一个测试类，这个测试类创建了两门课程，并向课程中加入学生。

程序清单10-5 TestCourse.java

```
1 public class TestCourse {
2   public static void main(String[] args) {
3     Course course1 = new Course("Data Structures");
4     Course course2 = new Course("Database Systems");
5
6     course1.addStudent("Peter Jones");
7     course1.addStudent("Brian Smith");
8     course1.addStudent("Anne Kennedy");
9
10    course2.addStudent("Peter Jones");
11    course2.addStudent("Steve Smith");
12
13    System.out.println("Number of students in course1: "
14      + course1.getNumberOfStudents());
15    String[] students = course1.getStudents();
16    for (int i = 0; i < course1.getNumberOfStudents(); i++)
17      System.out.print(students[i] + ", ");
18
19    System.out.println();
20    System.out.print("Number of students in course2: "
21      + course2.getNumberOfStudents());
22  }
23 }
```

```
Number of students in course1: 3
Peter Jones, Brian Smith, Anne Kennedy,
Number of students in course2: 2
```

Course类在程序清单10-6中执行。它使用一个数组存储选该门课的学生。为简单起见，假设选课的人数最多为100。在第3行使用new String[100]创建数组。addStudent方法（第10行）向这个数组中添加学生。只要有新的学生加入课程，numberOfStudents就增加1（第12行）。getStudents方法返回这个数组。dropStudent方法（第27行）留作练习。

程序清单10-6 Course.java

```
1 public class Course {
2   private String courseName;
3   private String[] students = new String[100];
4   private int numberOfStudents;
5
6   public Course(String courseName) {
7     this.courseName = courseName;
8   }
9
10  public void addStudent(String student) {
11    students[numberOfStudents] = student;
12    numberOfStudents++;
13  }
14
15  public String[] getStudents() {
16    return students;
17  }
18
19  public int getNumberOfStudents() {
20    return numberOfStudents;
21  }
22
23  public String getCourseName() {
24    return courseName;
25  }
26
27  public void dropStudent(String student) {
28    // Left as an exercise in Exercise 10.9
29  }
30 }
```

在程序清单10-6中，数组的大小固定为100（第3行）。可以在练习题10.9中改进它，使数组尺寸可以自动增加。

创建一个Course对象时，就创建了一个数组对象。Course对象包含对数组的引用。简单地说，这个Course对象包含了这个数组。

用户可以创建一个Course，然后通过公共方法addStudent、dropStudent、getNumberOfStudents和getStudents来处理它。然而，用户不需要知道这些方法是如何实现的。Course类封装了内部的实现。该例使用一个数组存储学生。也可以使用不同的数据结构存储学生。只要公共方法的合约保持不变，那么使用Course类的程序也无须修改。

10.9 设计堆栈类

回顾一下，栈（stack）是一种以“后进先出”的方式存放数据的数据结构，如图10-10所示。

栈有很多应用。例如：编译器使用栈来处理方法的调用。当调用某个方法时，方法的参数和局部变量都被压入栈中。当一个方法调用另一个方法时，新方法的参数和局部变量被压入栈中。当方法完成它的工作，返回它的调用者时，从栈中释放与它相关的空间。

可以定一个类建模栈。为简单起见，假设该栈存储int数值。因此，命名这个栈类为StackOfIntegers。这个类的UML图如图10-11所示。

假设该类是可用的。在程序清单10-7中编写一个测试程序，它使用该类创建一个栈（第3行），其中

存储了0，1，2，…，9这10个整数（第6行），然后按逆序显示它们（第9行）。

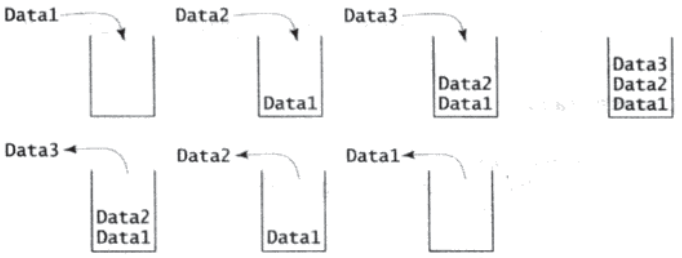


图10-10 栈是用后进先出的方式存放数据的

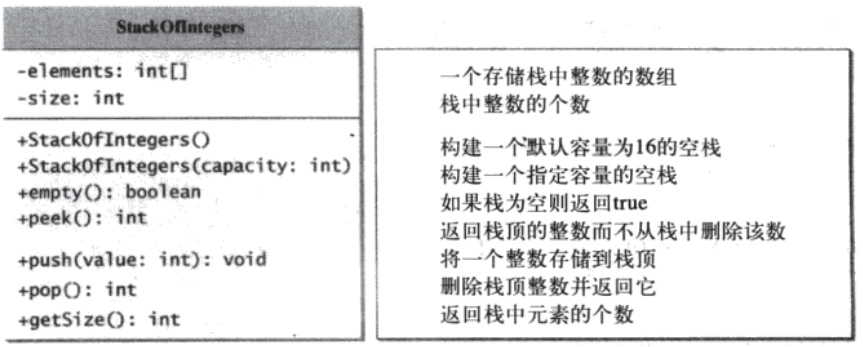


图10-11 StackOfIntegers类封装栈的存储，也提供处理栈的操作

程序清单10-7 TestStackOfIntegers.java

```
1 public class TestStackOfIntegers {
2   public static void main(String[] args) {
3     StackOfIntegers stack = new StackOfIntegers();
4
5     for (int i = 0; i < 10; i++)
6       stack.push(i);
7
8     while (!stack.empty())
9       System.out.print(stack.pop() + " ");
10  }
11 }
```

```
9 8 7 6 5 4 3 2 1 0
```

如何实现StackOfIntegers类呢？栈中的元素都存储在一个名为elements的数组中。创建一个栈的时候，同时也创建了这个数组。类的无参构造方法创建一个默认容量为16的数组。变量size记录了栈中元素的个数，而size–1是栈顶元素的下标，如图10-12所示。对空栈来说，size为0。

StackOfIntegers类是在程序清单10-8中实现的。方法empty()、peek()、pop()和getSize()都很容易实现。为了实现方法push(int value)，如果size<capacity，则将value赋值给elements[size]（第24行）。如果栈已满（即size>=capacity），则创建一个容量为当前容量两倍的新数组（第19行），将当前数组的内容复制到新数组中（第20行），并将新数组的引用赋值给栈中当前数组（第21行）。现在，可以给这个数组中添加新值（第24行）。

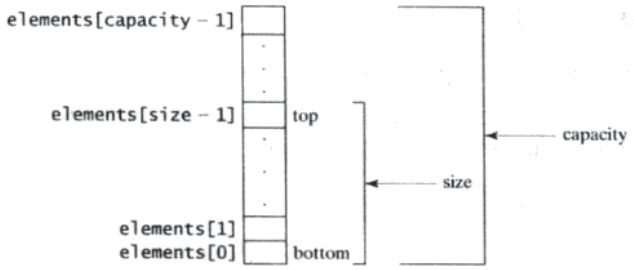


图10-12 StackOfIntegers类封装栈的存储，也提供处理栈的操作

程序清单10-8 StackOfIntegers.java

```
1 public class StackOfIntegers {
2   private int[] elements;
3   private int size;
4   public static final int DEFAULT_CAPACITY = 16;
5
6   /** Construct a stack with the default capacity 16 */
7   public StackOfIntegers() {
8     this(DEFAULT_CAPACITY);
9   }
10
11  /** Construct a stack with the specified maximum capacity */
12  public StackOfIntegers(int capacity) {
13    elements = new int[capacity];
14  }
15
16  /** Push a new integer into the top of the stack */
17  public void push(int value) {
18    if (size >= elements.length) {
19      int[] temp = new int[elements.length * 2];
20      System.arraycopy(elements, 0, temp, 0, elements.length);
21      elements = temp;
22    }
23
24    elements[size++] = value;
25  }
26
27  /** Return and remove the top element from the stack */
28  public int pop() {
29    return elements[--size];
30  }
31
32  /** Return the top element from the stack */
33  public int peek() {
34    return elements[size - 1];
35  }
36
37  /** Test whether the stack is empty */
38  public boolean empty() {
39    return size == 0;
40  }
41
42  /** Return the number of elements in the stack */
43  public int getSize() {
44    return size;
45  }
46 }
```

10.10 设计GuessDate类

程序清单3-3和程序清单7-6是两个猜测生日的程序。两个程序都使用相同的数据，并且都用面向过程范式开发。这两个程序的绝大多数代码就是定义五个数据集合。不能重用这两个程序的代码。为使代码可重用，需要设计一个如图10-13所定义的类来封装数据。

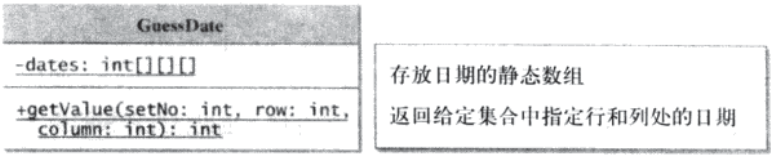


图10-13 GuessDate类定义猜测生日的数据

注意，getValue定义为一个静态方法，因为它不依赖于GuessDate类的某个特定对象。GuessDate类将dates封装为一个私有成员。这个类的使用者无须知道dates是如何实现的，甚至无须知道类中是否存在dates数据域。用户所需要知道的就是如何使用这些方法来访问数据。假设该类是可用的，如3.5节所示，这里有五个数据集合。调用getValue(setNo,row,column)会返回在给定集合中指定行和列的数据。例如：getValue(1,0,0)返回2。

假设GuessDate类是可用的。程序清单10-9是一个使用该类的测试程序。

程序清单10-9 UseGuessDateClass.java

```
1 import java.util.Scanner;
2
3 public class UseGuessDateClass {
4   public static void main(String[] args) {
5     int date = 0; // Date to be determined
6     int answer;
7
8     // Create a Scanner
9     Scanner input = new Scanner(System.in);
10
11    for (int i = 0; i < 5; i++) {
12      System.out.println("Is your birthday in Set" + (i + 1) + "?");
13      for (int j = 0; j < 4; j++) {
14        for (int k = 0; k < 4; k++)
15          System.out.print(GuessDate.getValue(i, j, k) + "  ");
16        System.out.println();
17      }
18
19      System.out.print("\nEnter 0 for No and 1 for Yes: ");
20      answer = input.nextInt();
21
22      if (answer == 1)
23        date += GuessDate.getValue(i, 0, 0);
24    }
25
26    System.out.println("Your birthday is " + date);
27  }
28 }
```

```
Is your birthday in Set1?
1  3  5  7
9  11  13  15
17  19  21  23
25  27  29  31
Enter 0 for No and 1 for Yes: 0 Enter
```

```
Is your birthday in Set2?
2  3  6  7
10  11  14  15
18  19  22  23
26  27  30  31
Enter 0 for No and 1 for Yes: 1 Enter

Is your birthday in Set3?
4  5  6  7
12  13  14  15
20  21  22  23
28  29  30  31
Enter 0 for No and 1 for Yes: 0 Enter

Is your birthday in Set4?
8  9  10  11
12  13  14  15
24  25  26  27
28  29  30  31
Enter 0 for No and 1 for Yes: 1 Enter

Is your birthday in Set5?
16  17  18  19
20  21  22  23
24  25  26  27
28  29  30  31
Enter 0 for No and 1 for Yes: 1 Enter

Your birthday is 26
```

因为getValue是一个静态方法，所以为了调用它，无须创建一个对象。GuessDate.getValue(i,j,k)（第15行）返回集合i中第i行第k列的数据。

GuessDate类可以在程序清单10-10中实现。

程序清单10-10 GuessDate.java

```
1 public class GuessDate {
2   private final static int[][][] dates = {
3     {{ 1,  3,  5,  7},
4      { 9, 11, 13, 15},
5      {17, 19, 21, 23},
6      {25, 27, 29, 31}},
7     {{ 2,  3,  6,  7},
8      {10, 11, 14, 15},
9      {18, 19, 22, 23},
10     {26, 27, 30, 31}},
11    {{ 4,  5,  6,  7},
12     {12, 13, 14, 15},
13     {20, 21, 22, 23},
14     {28, 29, 30, 31}},
15    {{ 8,  9, 10, 11},
16     {12, 13, 14, 15},
17     {24, 25, 26, 27},
18     {28, 29, 30, 31}},
19    {{16, 17, 18, 19},
20     {20, 21, 22, 23},
21     {24, 25, 26, 27},
22     {28, 29, 30, 31}}};
23
24   /** Prevent the user from creating objects from GuessDate */
25   private GuessDate() {
26   }
27
```

```
28   /** Return a date at the specified row and column in a given set */
29   public static int getValue(int setNo, int k, int j) {
30     return dates[setNo][k][j];
31   }
32 }
```

这个类使用一个三维数组存储数据（第2～22行）。可以使用不同的数据结构（即五个二维数组表示五个数字集合）。getValue方法的实现将被改变，但是使用GuessDate的程序无须改动，只要公共方法getValue的合约保持不变。这一点显示出数据封装的优点。

类定义了一个私有的无参构造方法（第25行），防止用户创建这个类的对象。由于这个类中的所有方法都是静态的，因此，无须创建这个类的对象。

10.11 类的设计原则

从前面两个例子以及前面几章中的其他许多例子中，已经学习了如何设计类。下面是一些设计原则。

10.11.1 内聚性

类应该描述一个单一的实体，而所有的类操作应该在逻辑上相互配合，支持一个连贯性的目标。例如：可以专门为学生使用一个类，但不应该将学生与教职工组合在同一个类中，因为学生和教职工是不同的实体。

如果一个实体担负太多的职责，就应该按各自的职责分成几个类。例如：String类、StringBuffer类和StringBuilder类都用于处理字符串，但是它们的职责不同。String类处理不可变字符串，StringBuilder类创建可变字符串，StringBuffer与StringBuilder类似，只是StringBuffer类还包含更新字符串的同步方法。

10.11.2 一致性

遵循标准Java程序设计风格和命名习惯。给类、数据域和方法选择有信息量的名字。流行的风格是将数据声明置于构造方法之前，并且将构造方法置于方法之前。

选择名字要保持一致。给类似的操作选择不同的名字并非好的习惯。例如：length()方法返回String、StringBuilder和StringBuffer的大小。如果在这些类中给这个方法用不同的名字就不一致了。

一般来说，应该一律提供一个为构造默认实例的公共无参构造方法。如果一个类不支持无参构造方法，要用文档写出原因。如果没有显式定义构造方法，就假定有一个空方法体的公共默认无参构造方法。

如果不想让用户创建类的对象，可以在类中声明一个私有的构造方法，就像Math类和GuessDate类的情况一样。

10.11.3 封装性

一个类应该使用private修饰符隐藏其数据，以免用户直接访问它。这使得类更易于维护。

如果想让数据域可读，只需提供get方法。如果想让数据域可更新，应该提供set方法。例如：Course类为courseName提供了get方法，但是没有提供set方法，这是因为一旦创建了课程类的对象，是不允许用户修改课程名字的。

10.11.4 清晰性

为使设计清晰，内聚性、一致性和封装性都是很好的设计原则。除此之外，类应该有一个很清晰的合约，易于解释和理解。

用户可以以多种不同组合和顺序，在许多不同环境中结合多个类。因此，应该设计一个类，这个类应该没有对用户使用目的及使用时间的限制，设计属性以容许用户按值的任何顺序和任何组合来设置，所设计的方法实现的功能应该与它们出现的顺序无关。例如：Loan类包含属性loanAmount、

numberOfYears和annualInterestRate。这些属性的值可以按任何顺序来设置。

方法应在不产生混淆的情况下凭直觉来定义。例如：String类中的substring(int beginIndex,int endIndex)方法就有一点混乱。这个方法返回从beginIndex到endIndex-1而不是endIndex的子串。

不应该声明一个来自其他数据域的数据域。例如，下面的Person类有两个数据域：birthDate和age。由于age可以从birthDate导出，所以age不应该声明为数据域。

```
public class Person {
  private java.util.Date birthDate;
  private int age;

  ...
}
```

10.11.5 完整性

类经常是为许多不同用户的使用而设计的。为了能在一个广泛的应用中使用，一个类应该通过属性和方法提供多种方案以适应用户的不同需求。例如：为满足不同的应用需求，String类包含了40多种很实用的方法。

10.11.6 实例和静态

依赖于类的具体实例的变量或方法应该是一个实例变量或方法。如果一个变量被类的所有实例所共享，那就应该将它声明为静态的。例如：在程序清单8-9中，Circle3中的变量numberOfObjects被SimpleCircle1类的所有对象共享。因此，它被声明为静态的。如果方法不依赖于某个具体的实例，那就应该将它声明为静态方法。例如：Circle3中的getNumberOfObjects方法都未绑定到任何具体实例，因此，它被声明为静态方法。

经常用类名（而不是引用变量）引用静态变量和方法，以增强可读性并避免错误。

不要从构造方法中传入参数来初始化静态数据域。最好使用set方法改变静态数据域。图a中的类最好用图b代替。

```
public class SomeThing {
  private int t1;
  private static int t2;

  public SomeThing(int t1, int t2) {
    ...
  }
}
```

a)

```
public class SomeThing {
  private int t1;
  private static int t2;

  public SomeThing(int t1) {
    ...
  }

  public static void setT2(int t2) {
    SomeThing.t2 = t2;
  }
}
```

b)

实例和静态是面向对象程序设计不可或缺的部分。数据域或方法要不是实例的就是静态的。不要错误地忽视了静态数据域或方法。常见的设计错误是将本应该声明为静态方法的方法声明为实例方法。例如：用于计算n的阶乘的factorial(int n)方法应该定义为静态的，因为它不依赖于任何具体实例。

构造方法永远都是实例方法，因为它是用来创建具体实例的。一个静态变量或方法可以从实例方法中调用，但是不能从静态方法中调用实例变量或方法。

关键术语

class abstraction（类抽象）　　class encapsulation（类封装）

class's contract（类的合约）　　class's variable（类变量）

immutable class（不可变类）　　stack（栈）
immutable object（不可变对象）　　this keyword（this关键字）

本章小结

- 不可变对象一旦创建，就不能改变。为防止用户修改对象，可以定义不可变类。
- 实例变量和静态变量的作用域是整个类，与变量在何处声明无关。实例变量和静态变量可以在类中的任意位置声明。为保持一致，最好在类的起始位置声明它们。
- 关键字this可用于表明调用对象。关键字this也可以用在构造方法中，调用同一个类的另一个构造方法。
- 面向过程范式重在设计方法。面向对象范式将数据和方法耦合在对象中。使用面向对象范式的软件设计重在对象和对象上的操作。面向对象方法结合了面向过程范式的功能以及将数据和操作集成在对象中的特点。

复习题

10.2节

10.1 如果一个类只有私有数据域而没有set方法，那么该类是不可变的吗？

10.2 如果一个类中所有的数据域都是私有的且都是基本类型，而且这个类也不包含set方法，那么该类是否是不可变的？

10.3 下面的类是不可变的吗？

```
public class A {
  private int[] values;

  public int[] getValues() {
    return values;
  }
}
```

10.4 如果无须set方法重新定义程序清单10-2中的Loan类，这个类是不可变的吗？

10.3节

10.5 下面程序的输出是什么？

```
public class Foo {
  private static int i = 0;
  private static int j = 0;

  public static void main(String[] args) {
    int i = 2;
    int k = 3;

    {
      int j = 3;
      System.out.println("i + j is " + i + j);
    }

    k = i + j;
    System.out.println("k is " + k);
    System.out.println("j is " + j);
  }
}
```

10.4节

10.6 描述this关键字的作用。下面代码有什么错误？

```
1 public class C {
2   private int p;
3
4   public C() {
5     System.out.println("C's no-arg constructor invoked");
6     this(0);
7   }
8
9   public C(int p) {
10    p = p;
11  }
12
13  public void setP(int p) {
14    p = p;
15  }
16 }
```

编程练习题

*10.1 （时间类Time）设计一个名为Time的类。这个类包含：

- 表示时间的数据域hour、minute和second。
- 一个以当前时间创建Time对象的无参构造方法（数据域的值表示当前时间）。
- 一个构造Time对象的构造方法，这个对象有一个特定的时间值，这个值是以毫秒表示的、从1970年1月1日午夜开始到现在流逝的时间段（数据域的值表示这个时间）。
- 一个构造带特定的小时、分钟和秒的Time对象的构造方法。
- 三个数据域hour、minute和second各自的get方法。
- 一个名为setTime(long elapseTime)的方法使用流逝的时间给对象设置一个新时间。

画出该类的UML图。实现这个类。编写一个测试程序，创建两个Time对象（使用new Time()和new Time(555550000)），然后显示它们的小时、分钟和秒。

提示 前两个构造方法可以从流逝的时间中提取出小时、分钟和秒。例如，如果逝去的时间为555550秒，那么转换为小时10、分钟19、秒数9。对于无参构造方法，当前时间可以使用System.currentTimeMills()获取当前时间，如程序清单2-6所示。

10.2 （BMI类）将下面的新构造方法加入到BMI类中：

```
/** Construct a BMI with the specified name, age, weight,
 * feet and inches
 */
public BMI(String name, int age, double weight, double feet,
double inches)
```

10.3 （MyInteger类）设计一个名为MyInteger的类。这个类包括：

- 一个名为value的int型数据域，存储这个对象表示的int值。
- 一个为指定的int值创建MyInteger对象的构造方法。
- 一个返回int值的get方法。
- 如果值分别为偶数、奇数或素数，那么isEven()、isOdd()和isPrime()方法都会返回true。
- 如果指定值分别为偶数、奇数或素数，那么isEven(int)、isOdd(int)和isPrime(int)方法都会返回true。

- 如果指定值分别为偶数、奇数或素数，那么isEven(MyInteger)、isOdd(MyInteger)和isPrime(MyInteger)方法都会返回true。
- 如果该对象的值与指定的值相等，那么equals(int)和equals(MyInteger)方法返回true。
- 静态方法parseInt(char[])将数字字符构成的数组转换为一个int值。
- 静态方法parseInt(String)将一个字符串转换为一个int值。

画出该类的UML图。实现这个类。编写客户程序测试这个类中的所有方法。

10.4 （MyPoint类）设计一个名为MyPoint的类，表示一个带x坐标和y坐标的点。该类包括：

- 两个带get方法的数据域x和y，分别表示它们的坐标。
- 一个创建点（0,0）的无参构造方法。
- 一个创建特定坐标点的构造方法。
- 两个数据域x和y各自的get方法。
- 一个名为distance的方法，返回MyPoint类型的两个点之间的距离。
- 一个名为distance的方法，返回指定x和y坐标的两个点之间的距离。

画出该类的UML图。实现这个类。编写一个测试程序，创建两个点（0,0）和（10,30.5），并显示它们之间的距离。

*10.5 （显示素数因子）编写一个程序，提示用户输入一个正整数，然后以降序显示它的所有最小因子。例如：如果整数为120，那么显示的最小因子为5、3、2、2、2。使用StackOfIntegers类存储因子（例如：2，2，2，3，5），获取之后按倒序显示这些因子。

*10.6 （显示素数）编写一个程序，然后按降序显示小于120的所有素数。使用StackOfIntegers类存储这些素数（例如：2，3，5，...），获取之后按倒序显示它们。

**10.7 （游戏：ATM机）使用练习题8.7中创建的Account类来模拟一台ATM机。创建一个有10个账户的数组，其id为0，1，…，9，并初始化收支为100美元。系统提示用户输入一个id。如果输入的id不正确，就要求用户输入正确的id。一旦接受一个id，就显示如运行示例所示的主菜单。可以选择1来查看当前的收支，选择2表示取钱，选择3表示存钱，选择4表示退出主菜单。一旦退出，系统就会提示再次输入id。所以，系统一旦启动，就不会停止。

```
Enter an id: 4 Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 Enter
The balance is 100.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 2 Enter
Enter an amount to withdraw: 3 Enter
```

```
Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 Enter
The balance is 97.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 3 Enter
Enter an amount to deposit: 10 Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 Enter
The balance is 107.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4 Enter

Enter an id:
```

**10.8 （财务问题：税款类Tax）练习题7.12使用数组编写一个计算税款的程序。设计一个名为Tax的类，该类包含下面的实例数据域：

- int filingStatus，四种纳税人状态之一：0——单身纳税人、1——已婚共缴纳税人、2——已婚单缴纳税人、3——家庭纳税人。使用公共静态常量SINGLE_FILER(0)、MARRIED_JOINTLY(1)、MARRIED_SEPARATELY(2)和HEAD_OF_HOUSEHOLD(3)表示这些状态。
- int[][] brackets：存储每种纳税人的纳税等级。
- double[] rates：存储每种纳税等级的税率。
- double taxableIncome：存储可征税收入。

给每个数据域提供get和set方法，并提供返回税款的getTax()方法。该类还提供一个无参构造方法和构造方法Tax(filingStatus,brackets,rates,taxableIncome)。

画出该类的UML图。实现这个类。编写一个测试程序，使用Tax类对所给四种纳税人打印2001年和2009年的税款表，可征税收入范围在50 000美元和60 000美元之间，间隔区间为1000美元。2009年的税率参见表3-2，2001年的税率参见表10-1。

表10-1 2001年美国联邦个人所得税税率表

税 率	单身纳税人	已婚共缴纳税人或合法寡妇	已婚单缴纳税人	户主纳税人
15%	$27 050以下	$45 200以下	$22 600以下	$36 250以下
27.5%	$27 051～$65 550	$45 201～$109 250	$22 601～$54 625	$36 251-$93 650
30.5%	$65 551～$136 750	$109 251～166 500	$54 626～$83 250	$93 651-$151 650
35.5%	$136 751～$29 7350	$166 501～$297 350	$83 251～$148 675	$151 651-$297 350
39.1%	$297 351及以上	$297 351及以上	$148 676及以上	$297 351及以上

**10.9 （课程类Course）如下改写Course类：

- 程序清单10-6中数组的大小是固定的。对它进行改进，通过创建一个新的更大的数组并复制当前数组的内容来实现数组大小的自动增长。
- 实现dropStudent方法。
- 添加一个名为clear()的新方法，然后删掉选某门课程的所有学生。

编写一个测试程序，创建一门课程，添加三个学生，删掉一个学生，然后显示这门课程的学生。

*10.10 （游戏：GuessDate类）修改程序清单10-10中的GuessDate类。不使用三维数组表示数据，而是用五个二维数组来表示五个集合的数字。所以，需要声明：

```
private static int[][] set1 = {{1,  3,  5,  7}, ... };
private static int[][] set2 = {{2,  3,  6,  7}, ... };
private static int[][] set3 = {{4,  5,  6,  7}, ... };
private static int[][] set4 = {{8,  9, 10, 11}, ... };
private static int[][] set5 = {{16, 17, 18, 19}, ... };
```

*10.11 （几何方面：Circle2D类）定义Circle2D类，包括：

- 两个带有get方法的名为x和y的double型数据域，表明圆的中心点。
- 一个带get方法的数据域radius。
- 一个无参构造方法，该方法创建一个（x,y）值为（0,0）且radius为1的默认圆。
- 一个构造方法，创建带指定的x，y和radius的圆。
- 一个返回圆面积的方法getArea()。
- 一个返回圆周长的方法getPerimeter()。
- 如果给定的点（x，y）在圆内，那么方法contains(double x，double y)返回true。如图10-14a所示。
- 如果给定的圆在这个圆内，那么方法contains(Circle2D circle)返回true。如图10-14b所示。
- 如果给定的圆和这个圆重叠，那么方法overlaps(Circle2D circle)返回true。如图10-14c所示。

画出该类的UML图。实现这个类。编写测试程序，创建一个Circle2D对象c1(new Circle2D(2,2,5.5))，显示它的面积和周长，还要显示c1.contains(3,3)、c1.contains(new Circle2D (4,5,10.5))和c1.overlaps(new Circle2D(3,5,2.3))。

图10-14 a）点在圆内；b）一个圆在另一个圆内；c）一个圆和另一个圆重叠

*10.12 （几何问题：MyRectangle2D类）定义MyRectangle2D类，包含：

- 两个名为x和y的double型数据域，表明矩形的中心点，这两个数据域都带有get和set方法（假设这个矩形的边与x轴和y轴平行）。
- 带get和set方法的数据域width和height。
- 一个无参构造方法，该方法创建一个（x，y）值为（0，0）且width和height为1的默认矩形。
- 一个构造方法，创建带指定的x、y、width和height的矩形。
- 方法getAreac()返回矩形的面积。
- 方法getPerimeter()返回矩形的周长。
- 如果给定的点（x，y）在矩形内，那么方法contains(double x，double y)返回true。如图10-15a所示。

- 如果给定的矩形在这个矩形内，那么方法contains(MyRectangle2D r)返回true。如图10-15b所示。
- 如果给定的矩形和这个矩形重叠，那么方法overlaps(MyRectangle2D r)返回true。如图10-15c所示。

画出该类的UML图。实现这个类。编写测试程序，创建一个MyRectangle2D对象r1(new MyRectangle2D(2,2,5.5,4.9))，显示它的面积和周长，然后显示r1.contains(3,3)、r1.contains(new MyRectangle2D(4,5,10.5, 3.2))和 r1.overlaps(new MyRectangle2D(3,5, 2.3,5.4))的结果。

a) b) c)

图10-15 a）点在矩形内；b）一个矩形在另一个矩形内；c）一个矩形和另一个矩形重叠

***10.13 （几何问题：Triangle2D类）定义Triangle2D类，包含：

- 三个名为p1、p2和p3的MyPoint型数据域，这三个数据域都带有get和set方法。MyPoint在练习题10.4中定义。
- 一个无参构造方法，该方法创建三个坐标为（0，0）、（1，1）和（2，5）的点组成的默认三角形。
- 一个创建带指定点的三角形的构造方法。
- 一个返回三角形面积的方法getArea()。
- 一个返回三角形周长的方法getPerimeter()。
- 如果给定的点p在这个三角形内，那么方法contains(MyPoint p)返回true。如图10-16a所示。
- 如果给定的三角形在这个三角形内，那么方法contains(Triangle2D t)返回true。如图10-16b所示。
- 如果给定的三角形和这个三角形重叠，那么方法overlaps(Triangle2D t)返回true。如图10-16c所示。

a) b) c)

图10-16 a）点在三角形内；b）一个三角形在另一个三角形内；c）一个三角形和另一个三角形重叠

画出该类的UML图。实现这个类。编写测试程序，创建一个Triangle2D对象t1(new Triangle2D(new MyPoint(2.5,2),new MyPoint(4.2,3),new MyPoint(5,3.5)))，显示它的面积和周长，还要显示t1.contains（3,3）、t1.contains(new Triangle2D(new MyPoint(2.9,2), new MyPoint(4,1), MyPoint(1,3.4)))和t1.overlaps(new Triangle2D(new MyPoint(2,5.5),new MyPoint(4,-3), MyPoint(2,6.5)))的结果。

提示 关于计算三角形面积的公式，请参见练习题5.19。使用Java API中的java.awt.geo.Line2D类来实现contains和overlaps方法。Line2D类包括检查两条线段是否相交以及检查一条线是否包含一个点等方法。请参见Java API获取关于Line2D更多的信息。为了检测一个点是否在三角形中，画三条虚线，如图10-17所示。如果点在三角形中，每条虚线应该和边相交一次。如果虚线和边相交两次，那么这个点肯定在这个三角形外。

a) b)

图10-17 a）点在三角形内；b）一个点在三角形外

*10.14 （MyDate类）设计一个名为MyDate类。该类包含：

- 表示日期的数据域year、month和day。
- 一个无参构造方法，该方法创建当前日期的MyDate对象。
- 一个构造方法，创建以从1970年1月1日午夜开始流逝的毫秒数为时间的MyDate对象。
- 一个构造方法，创建一个带指定年、月、日的MyDate对象。
- 三个数据域year、month和day各自的get方法。
- 一个名为setDate(long elapsedTime)使用流逝的时间为对象设置新数据的方法。

画出该类的UML图。实现这个类。编写测试程序，创建一个测试程序，创建两个Date对象（使用new Date()和new Date(34355555133101L)，然后显示它们的小时、分钟和秒。

提示 前两个构造方法将从逝去的时间中提取出年、月、日。例如：如果逝去的时间是561555550000毫秒，那么年就是1987，月就是10，而天是17。对于无参构造方法，当前时间可以使用System.currentTimeMills()获取，如程序清单2-6所示。