继承和多态

学习目标

- 通过继承由父类创建子类（11.2节）。
- 使用关键字`super`调用父类的构造方法和方法（11.3节）。
- 在子类中覆盖方法（11.4节）。
- 区分覆盖和重载的不同（11.5节）。
- 探究`Object`类中的`toString()`方法（11.6节）。
- 理解多态性和动态绑定（11.7～11.8节）。
- 描述转换并解释显式向下转换的必要性（11.9节）。
- 探究`Object`类中的`equals`方法（11.10节）。
- 存储、提取和操作`ArrayList`中的对象（11.11节）。
- 使用`ArrayList`实现`Stack`类（11.12节）。
- 使用可见性修饰符`protected`使父类中的数据和方法可以被子类访问（11.13节）。
- 使用修饰符`final`防止类的扩展以及方法的覆盖（11.14节）。

11.1 引言

在面向对象程序设计中，可以从已有的类派生出新类。这称做*继承*（inheritance）。继承是Java在软件重用方面一个重要且功能强大的特征。假设要定义一个类，对圆、矩形和三角形建模。这些类有很多共同的特性。设计这些类来避免冗余并使系统更易于理解和易于维护的最好的方式是什么？答案就是使用继承。

11.2 父类和子类

使用类来对同一类型的对象建模。不同的类也可能会有一些共同的特征和动作，这些共同的特征和行动都统一放在一个类中，它是可以被其他类所共享的。继承可以让你定义一个通用类，随后将它扩展为更多特定的类。这些特定的类继承通用类中的特征和方法。

考虑一下几何对象。假设要设计类建模像圆和矩形这样的几何对象。几何对象有许多共同的属性和行为。它们可以是用某种颜色画出来的、填充的或者不填充的。这样，一个通用类`GeometricObject`可以用来建模所有的几何对象。这个类包括属性`color`和`filled`，以及适用于这些属性的`get`和`set`方法。假设该类还包括`dateCreated`属性以及`getDateCreated()`和`toString()`方法。`toString()`方法返回代表该对象的字符串。由于圆是一个特殊类型的几何对象，所以它和其他几何对象共享共同的属性和方法。因此，通过扩展`GeometricObject`类来定义`Circle`类是很有意义的。同理，`Rectangle`也可以声明为`GeometricObject`的子类。图11-1显示这些类之间的关系。指向父类的箭头用来表示相关的两个类之间的继承关系。

在Java的术语中，如果类`C1`扩展自另一个类`C2`，那么就将`C1`称为*次类*（subclass），将`C2`称为*超类*（superclass）。超类也称为*父类*（parent class）或*基类*（base class），次类又称为*子类*（child class）、*扩展类*（extended class）或*派生类*（derived class）。子类从它的父类中继承可访问的数据域和方法，还可以添加新数据域和新方法。

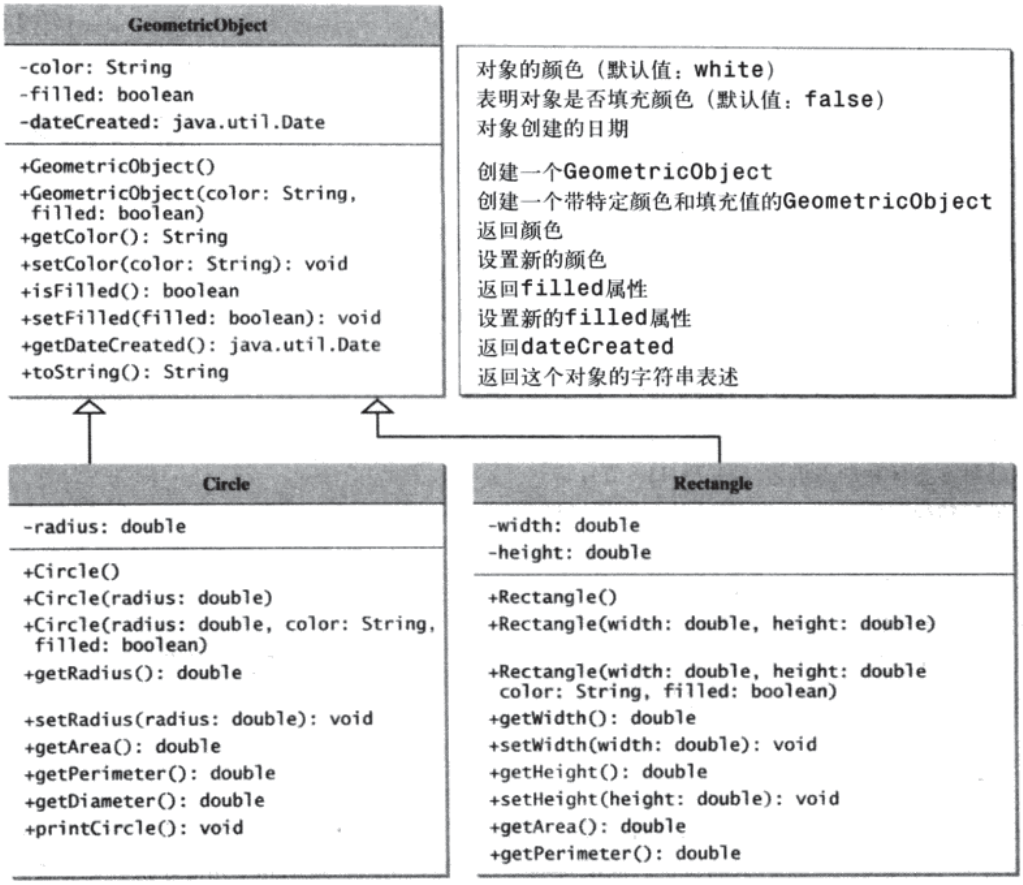


图11-1 GeometricObject类是Circle类和Rectangle类的父类

Circle类继承了GeometricObject类所有可以访问的数据域和方法。除此之外，它还有一个新的数据域radius，以及与radius相关的get和set方法。它还包括getArea()、getPerimeter()和getDiameter()方法以返回圆的面积、周长和直径。

Rectangle类从GeometricObject类继承所有可访问的数据域和方法。除此之外，它还有width和height数据域，以及和它们相关的get和set方法。它还包括getArea()和getPerimeter()方法返回矩形的面积和周长。

GeometricObject类、Circle类和Rectangle类分别显示在程序清单11-1、程序清单11-2和程序清单11-3中。

注意 为了避免与第12章介绍的改进版的GeometricObject类、Circle类和Rectangle类发生命名冲突，就将本章的类命名为GeometricObject1、Circle4和Rectangle1。为方便起见，在文字描述上仍称它们为GeometricObject、Circle和Rectangle类。避免命名冲突最好的方法应该是将这些类放到不同的包中。然而，为了简单性和一致性，本书中所有的类都放在某个默认的包内。

程序清单11-1 GeometricObject1.java

```
1 public class GeometricObject1 {
2   private String color = "white";
3   private boolean filled;
4   private java.util.Date dateCreated;
5
```

```
6   /** Construct a default geometric object */
7   public GeometricObject1() {
8     dateCreated = new java.util.Date();
9   }
10
11  /** Construct a geometric object with the specified color
12    * and filled value */
13  public GeometricObject1(String color, boolean filled) {
14    dateCreated = new java.util.Date();
15    this.color = color;
16    this.filled = filled;
17  }
18
19  /** Return color */
20  public String getColor() {
21    return color;
22  }
23
24  /** Set a new color */
25  public void setColor(String color) {
26    this.color = color;
27  }
28
29  /** Return filled. Since filled is boolean,
30     its get method is named isFilled */
31  public boolean isFilled() {
32    return filled;
33  }
34
35  /** Set a new filled */
36  public void setFilled(boolean filled) {
37    this.filled = filled;
38  }
39
40  /** Get dateCreated */
41  public java.util.Date getDateCreated() {
42    return dateCreated;
43  }
44
45  /** Return a string representation of this object */
46  public String toString() {
47    return "created on " + dateCreated + "\ncolor: " + color +
48      " and filled: " + filled;
49  }
50 }
```

程序清单11-2 Circle4.java

```
1 public class Circle4 extends GeometricObject1 {
2   private double radius;
3
4   public Circle4() {
5   }
6
7   public Circle4(double radius) {
8     this.radius = radius;
9   }
10
11  public Circle4(double radius, String color, boolean filled) {
12    this.radius = radius;
13    setColor(color);
14    setFilled(filled);
15  }
16
17  /** Return radius */
```

```
18   public double getRadius() {
19     return radius;
20   }
21
22   /** Set a new radius */
23   public void setRadius(double radius) {
24     this.radius = radius;
25   }
26
27   /** Return area */
28   public double getArea() {
29     return radius * radius * Math.PI;
30   }
31
32   /** Return diameter */
33   public double getDiameter() {
34     return 2 * radius;
35   }
36
37   /** Return perimeter */
38   public double getPerimeter() {
39     return 2 * radius * Math.PI;
40   }
41
42   /* Print the circle info */
43   public void printCircle() {
44     System.out.println("The circle is created " + getDateCreated() +
45       " and the radius is " + radius);
46   }
47 }
```

Circle类使用下面的语法扩展GeometricObject类（程序清单11-2）：

关键字extends（第1行）告诉编译器，Circle类扩展GeometricObject类，这样，它就继承了getColor、setColor、isFilled、setFilled和toString方法。

重载的构造方法Circle(double radius,String color,boolean filled)是通过调用getColor和setFilled方法设置color和filled属性来执行的（第11～15行）。这两个公共方法是在基类GeometricObject中定义的，在Circle中继承。因此，可以在派生类中使用它们。

可以尝试在构造方法中使用数据域color和filled，如下所示：

```
public Circle4(double radius, String color, boolean filled) {
  this.radius = radius;
  this.color = color; // Illegal
  this.filled = filled; // Illegal
}
```

这是错的，因为GeometricObject类中的私有数据域color和filled是不能被除了GeometricObject类本身之外的其他任何类访问的。唯一读取和改变color与filled的方法就是通过它们的get和set方法。

Rectangle类（程序清单11-3）使用下面的语法扩展GeometricObject类（程序清单11-2）：

关键字extends（第1行）告诉编译器Rectangle类扩展GeometricObject类，这样，它就继承了getColor、setColor、isFilled、setFilled和toString方法。

程序清单11-3 Rectangle1.java

```
1 public class Rectangle1 extends GeometricObject1 {
2   private double width;
3   private double height;
4
5   public Rectangle1() {
6   }
7
8   public Rectangle1(double width, double height) {
9     this.width = width;
10    this.height = height;
11  }
12
13  public Rectangle1(double width, double height, String color,
14      boolean filled) {
15    this.width = width;
16    this.height = height;
17    setColor(color);
18    setFilled(filled);
19  }
20
21  /** Return width */
22  public double getWidth() {
23    return width;
24  }
25
26  /** Set a new width */
27  public void setWidth(double width) {
28    this.width = width;
29  }
30
31  /** Return height */
32  public double getHeight() {
33    return height;
34  }
35
36  /** Set a new height */
37  public void setHeight(double height) {
38    this.height = height;
39  }
40
41  /** Return area */
42  public double getArea() {
43    return width * height;
44  }
45
46  /** Return perimeter */
47  public double getPerimeter() {
48    return 2 * (width + height);
49  }
50 }
```

程序清单11-4中的代码创建了Circle和Rectangle的对象，并调用这些对象上的方法。toString()方法继承自GeometricObject类，并且从Circle对象（第4行）和Rectangle对象（第10行）调用。

程序清单11-4 TestCircleRectangle.java

```
1 public class TestCircleRectangle {
2   public static void main(String[] args) {
3     Circle4 circle = new Circle4(1);
4     System.out.println("A circle " + circle.toString());
5     System.out.println("The radius is " + circle.getRadius());
```

```
    System.out.println("The area is " + circle.getArea());
    System.out.println("The diameter is " + circle.getDiameter());

    Rectangle1 rectangle = new Rectangle1(2, 4);
    System.out.println("\nA rectangle " + rectangle.toString());
    System.out.println("The area is " + rectangle.getArea());
    System.out.println("The perimeter is " +
      rectangle.getPerimeter());
  }
}
```

```
A circle created on Thu Sep 24 20:31:02 EDT 2009
color: white and filled: false
The radius is 1.0
The area is 3.141592653589793
The diameter is 2.0

A rectangle created on Thu Sep 24 20:31:02 EDT 2009
color: white and filled: false
The area is 8.0
The perimeter is 12.0
```

下面是在考虑继承时很值得注意的几点：

- 和传统的理解相反，子类并不是父类的一个子集。实际上，一个子类通常比它的父类包含更多的信息和方法。
- 父类中的私有数据域在该类之外是不可访问的。因此，不能在子类中直接使用它们。但是，如果父类中定义了公共的访问器/修改器，那么可以通过这些公共的访问器/修改器来访问和修改它们。
- 不是所有的是关系（is-a）都该用继承来建模。例如：一个正方形是一个矩形，但是不应该定义一个Square类来扩展Rectangle类，因为没有任何东西可以从矩形扩展（或者补充）到正方形。所以，应该定义一个扩展自GeometricObject类的Square类。如果要用类B去扩展类A，那么A应该要比B包含更多的信息。
- 继承是用来为是关系（is-a）建模的。不要仅仅为了重用方法这个原因而盲目地扩展一个类。例如：尽管Person类和Tree类可以共享类似高度和重量这样的通用特性，但是从Person类扩展出Tree类是毫无意义的。一个父类和它的子类之间必须存在是关系。
- 某些程序设计语言是允许从几个类派生出一个子类的。这种能力称为多重继承（multiple inheritance）。但是在Java中，是不允许多重继承的。一个Java类只可能直接继承自一个父类。这种限制称为单一继承（single inheritance）。如果使用extends关键字来定义一个子类，它只允许有一个父类。然而，多重继承是可以通过接口来实现的，这部分内容将在14.4节中介绍。

11.3 使用super关键字

子类继承它的父类中所有可访问的数据域和方法。它能继承构造方法吗？父类的构造方法能够从子类调用吗？本节就来解决这些问题以及同它们相关的问题。

10.4节中介绍了关键字this的作用，它是一个调用对象的引用。关键字super是指这个super出现的类的父类。关键字super可以用于两种途径：

1）调用父类的构造方法。

2）调用父类的方法。

11.3.1 调用父类的构造方法

调用父类构造方法的语法是：

```
super(), or super(parameters);
```

语句super()调用它的父类的无参构造方法，而语句super（参数）调用与参数匹配的父类的构造方法。语句super()和super（参数）必须出现在子类构造方法的第一行，这是显式调用父类构造方法的唯一方式。例如，在程序清单11-2中的第11～15行的构造方法可以用下面的代码替换：

```
public Circle4(double radius, String color, boolean filled) {
  super(color, filled);
  this.radius = radius;
}
```

警告 要调用父类构造方法就必须使用关键字super，而且这个调用必须是构造方法的第一条语句。在子类中调用父类构造方法的名字会引起一个语法错误。

注意 构造方法可用来构造一个类的实例。不像属性和方法，父类的构造方法是不被子类继承的。它们只能从子类的构造方法中用关键字super调用。

11.3.2 构造方法链

构造方法可以调用重载的构造方法或它的父类的构造方法。如果它们都没有被显式地调用，编译器就会自动地将super()作为构造方法的第一条语句。例如：

```
public ClassName() {
  // some statements
}
```

等价于

```
public ClassName() {
  super();
  // some statements
}
```

```
public ClassName(double d) {
  // some statements
}
```

等价于

```
public ClassName(double d) {
  super();
  // some statements
}
```

在任何情况下，构造一个类的实例时，将会调用沿着继承链的所有父类的构造方法。当构造一个子类的对象时，子类构造方法会在完成自己的任务之前，首先调用它的父类的构造方法。如果父类继承自其他类，那么父类构造方法又会在完成自己的任务之前，调用它自己的父类的构造方法。这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止。这就是构造方法链（constructor chaining）。

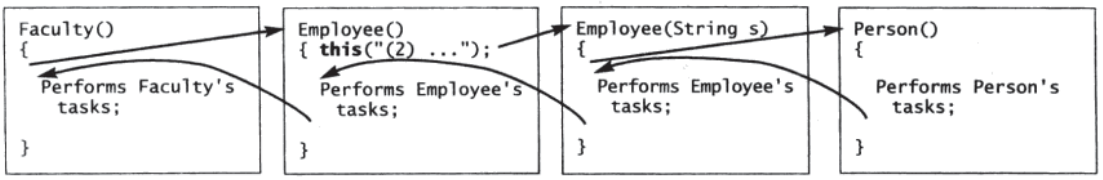
考虑下面的代码：

```
1 public class Faculty extends Employee {
2   public static void main(String[] args) {
3     new Faculty();
4   }
5
6   public Faculty() {
7     System.out.println("(4) Performs Faculty's tasks");
8   }
9 }
10
11 class Employee extends Person {
12   public Employee() {
13     this("(2) Invoke Employee's overloaded constructor");
14     System.out.println("(3) Performs Employee's tasks ");
15   }
16
17   public Employee(String s) {
18     System.out.println(s);
19   }
20 }
21
22 class Person {
23   public Person() {
```

```
24      System.out.println("(1) Performs Person's tasks");
25    }
26 }
```

```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```

该程序会产生上面的输出。为什么呢？我们讨论一下这个原因。在第3行，new Faculty()调用Faculty的无参构造方法。由于Faculty是Employee的子类，所以，在Faculty构造方法中的所有语句执行之前，先调用Employee的无参构造方法。Employee的无参构造方法调用Employee的第二个构造方法（第12行）。由于Employee是Person的子类，所以，在Employee的第二个构造方法中所有语句执行之前，先调用Person的无参构造方法。这个过程如下图所示：



警告 如果一个类要设计为扩展的，最好提供一个无参构造方法以避免程序设计错误。考虑下面的代码：

```
1 public class Apple extends Fruit  {
2 }
3
4 class Fruit  {
5   public Fruit(String name) {
6     System.out.println("Fruit's constructor is invoked");
7   }
8 }
```

由于在Apple中没有显式定义的构造方法，因此，Apple的默认无参构造方法被隐式调用。因为Apple是Fruit的子类，所以Apple的默认构造方法会自动调用Fruit的无参构造方法。然而，Fruit没有无参构造方法，因为Fruit显式地定义了构造方法。因此，程序不能被编译。

设计指南 最好能为每个类提供一个无参构造方法（如果需要的话），以便于对该类进行扩展同时避免错误。

11.3.3 调用父类的方法

关键字super不仅可以引用父类的构造方法，也可以引用父类的方法。所用语法如下：

super.方法名（参数）；

可以如下改写Circle类中的printCircle()方法：

```
public void printCircle() {
  System.out.println("The circle is created " +
    super.getDateCreated() + " and the radius is " + radius);
}
```

然而，在这种情况下，没有必要在getDateCreated()前放置super，因为getDateCreated是GeometricObject类中的一个方法，可以被Circle类继承。然而，在有些情况下，如11.4节所示，关键字super是必不可少的。

11.4 覆盖方法

子类从父类中继承方法。有时，子类需要修改父类中定义的方法的实现，这称做*方法覆盖*（method overriding）。

`GeometricObject`类中的`toString`方法返回表示几何对象的字符串。这个方法可以被覆盖，返回表示圆的字符串。为了覆盖它，在程序清单11-2中加入下面的新方法：

```
1 public class Circle4 extends GeometricObject1 {
2   // Other methods are omitted
3
4   /** Override the toString method defined in GeometricObject */
5   public String toString() {
6     return super.toString() + "\nradius is " + radius;
7   }
8 }
```

`toString()`方法在`GeometricObject`类中定义，在`Circle`类中修改。在这两个类中定义的该方法都可以在`Circle`类中使用。要在`Circle`类中调用定义在`GeometricObject`中的`toString`方法，使用`super.toString()`（第6行）。

`Circle`的子类能用语法`super.super.toString()`访问定义在`GeometricObject`中的`toString`方法吗？答案是不能。这是一个语法错误。

以下几点值得注意：

- 仅当实例方法是可访问时，它才能被覆盖。这样，因为私有方法在它的类本身以外是不能访问的，所以它不能被覆盖。如果子类中定义的方法在父类中是私有的，那么这两个方法完全没有关系。
- 与实例方法一样，静态方法也能被继承。但是，静态方法不能被覆盖。如果父类中定义的静态方法在子类中被重新定义，那么定义在父类中的静态方法将被隐藏。可以使用语法：父类名.静态方法名（`SuperClassName.staticMethodName`）调用隐藏的静态方法。

11.5 覆盖和重载

在5.8节中已经学过关于重载方法的内容。重载方法意味着可以定义多个同名的方法，但这些方法具有不同的签名。覆盖方法意味着为子类中的方法提供一个全新的实现。该方法已经在父类中定义。

为了覆盖一个方法，这个方法必须使用相同的签名以及相同的返回值类型在子类中进行定义。

用一个例子来显示覆盖和重载的不同。在图a中，类A中的方法`p(double i)`覆盖了在类B中定义的同一个方法。但是，在图b中，类B中有两个重载的方法`p(double i)`和`p(int i)`。类B的`p(double i)`方法被继承。

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

a)

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

b)

运行图a中的Test类时，a.p(10)和a.p(10.0)调用的都是定义在类A中的p(double i)方法，所以程序都显示10.0。运行图b中的Test类时，a.p(10)调用类A中定义的p(int i)方法，显示输出为10，而a.p(10.0)调用定义在类B中的p(double i)方法，显示输出为20.0。

11.6 对象类Object和它的toString()方法

Java中的每个类都源于java.lang.Object类。如果在定义一个类时没有指定继承性，那么这个类的父类就被默认为是Object。例如，下面两个类的定义是一样的：

```
public class ClassName {
  ...
}
```

等价于

```
public class ClassName extends Object {
  ...
}
```

诸如String、StringBuilder、Loan和GeometricObject这样的类都是Object的隐含子类（此前在本书中见到的所有主类也是如此）。熟悉Object类提供的方法是非常重要的，因为这样就可以在自己的类中使用它们。本节将介绍Object类中的toString()方法。

toString()方法的签名是：

```
public String toString()
```

调用一个对象的toString()会返回一个描述该对象的字符串。默认情况下，它返回一个由该对象所属的类名、at符号（@）以及该对象十六进制形式的内存地址组成的字符串。例如，考虑下面这些在程序清单10-2中定义Loan类的代码：

```
Loan loan = new Loan();
System.out.println(loan.toString());
```

这些代码会显示像Loan@15037e5的字符串。这个信息不是很有用，或者说没有什么信息量。通常，应该覆盖这个toString方法，这样，它可以返回一个代表该对象的描述性字符串。例如，Object类中的toString方法在GeometricObject类中覆盖，如程序清单11-1中第46～49行所示：

```
public String toString() {
  return "created on " + dateCreated + "\ncolor: " + color +
    " and filled: " + filled;
}
```

注意 也可以传递一个对象来调用System.out.println(object)或者System.out.print(object)。这等价于调用System.out.println(object.toString())或者System.out.print (object.toString())。因此，应该使用System.out.println(loan)来替换System.out.println(loan.toString())。

11.7 多态

面向对象程序设计的三个特点是封装、继承和多态。前面已经学习了前两个特点，本节将介绍多态性。

首先，定义两个有用的术语：子类型和父类型。一个类实际上定义了一种类型。子类定义的类型称为子类型（subtype），而父类定义的类型称为父类型（supertype）。因此，可以说Circle是GeometricObject的子类型，而GeometricObject是Circle的父类型。

继承关系使一个子类继承父类的特征，并且附加一些新特征。子类是它的父类的特殊化，每个子类的实例都是其父类的实例，但是反过来就不成立。例如：每个圆都是一个几何对象，但并非每个几何对象都是圆。因此，总可以将子类的实例传给需要父类类型的参数。考虑程序清单11-5中的代码。

程序清单11-5 PolymorphismDemo.java

```
1 public class PolymorphismDemo {
2   /** Main method */
3   public static void main(String[] args) {
4     // Display circle and rectangle properties
5     displayObject(new Circle4(1, "red", false));
6     displayObject(new Rectangle1(1, 1, "black", true));
7   }
8
9   /** Display geometric object properties */
10  public static void displayObject(GeometricObject1 object) {
11    System.out.println("Created on " + object.getDateCreated() +
12      ". Color is " + object.getColor());
13  }
14 }
```

```
Created on Mon Mar 09 19:25:20 EDT 2009. Color is white
Created on Mon Mar 09 19:25:20 EDT 2009. Color is black
```

方法displayObject（第10行）采用GeometricObject类型的参数。可以通过传递任何一个GeometricObject的实例（例如：在第5～6行的new Circle4(1,"red",false)和new Rectangle1(1,1,"black",false))来调用displayObject。使用父类对象的地方都可以使用子类的对象。这就是通常所说的多态（polymorphism，它源于希腊文字，意思是“多种形式”）。用简单的术语来描述，多态就意味着父类型的变量可以引用子类型的对象。

11.8 动态绑定

一个方法可以在父类中定义而在它的子类中覆盖。例如：toString()方法是在Object类中定义的，而该方法在GeometricObject1类中覆盖。考虑下面的代码：

```
Object o = new GeometricObject();
System.out.println(o.toString());
```

这里的o调用哪个toString()呢？为了回答这个问题，我们首先介绍两个术语：声明类型和实际类型。一个变量必须被声明为某种类型。变量的这个类型称为它的*声明类型*（declared type）。这里，o的声明类型是Object。一个引用类型变量可以是一个null值或者是一个对声明类型实例的引用。实例可以使用声明类型或它的子类型的构造方法创建。变量的*实际类型*（actual type）是被变量引用的对象的实际类。这里，o的实际类型是GeometricObject，因为o指向使用new GeometricObject()创建的对象。o调用的到底是哪个toString()方法是由o的实际类型所决定的。这称为*动态绑定*（dynamic binding）。

动态绑定工作机制如下：假设对象o是类C_1，C_2，…，C_{n-1}，C_n的实例，其中C_1是C_2的子类，C_2是C_3的子类，…，C_{n-1}是C_n的子类，如图11-2所示。也就是说，C_n是最通用的类，C_1是最特殊的类。在Java中，C_n是Object类。如果对象o调用一个方法p，那么Java虚拟机会依次在类C_1，C_2，…，C_{n-1}，C_n中查找方法p的实现，直到找到为止。一旦找到一个实现，就停止查找然后调用这个第一次找到的实现。

图11-2 将调用的方法是在运行时动态决定的

程序清单11-6给出一个演示动态绑定的例子。

程序清单11-6 DynamicBindingDemo.java

```
1 public class DynamicBindingDemo {
2   public static void main(String[] args) {
3     m(new GraduateStudent());
4     m(new Student());
5     m(new Person());
6     m(new Object());
7   }
8
9   public static void m(Object x) {
10    System.out.println(x.toString());
11  }
12 }
13
14 class GraduateStudent extends Student {
15 }
16
17 class Student extends Person {
18   public String toString() {
19     return "Student";
20   }
21 }
22
23 class Person extends Object {
24   public String toString() {
25     return "Person";
26   }
27 }
```

```
Student
Student
Person
java.lang.Object@130c19b
```

方法m（第9行）采用Object类型的参数。可以用任何对象（例如：在第3～6行的new GraduateStudent()、new Student()、new Person()和new Object()）作为参数来调用m方法。

当执行方法m(Object x)时，调用参数x的toString方法。x可能会是GraduateStudent、Student、Person或者Object的实例。类GraduateStudent、Student、Person以及Object都有它们自己对toString方法的实现。使用哪个实现取决于运行时x的实际类型。调用m(new GraduateStudent())（第3行）会导致定义在Student类中的toString方法被调用。

调用m(new Student())（第4行）会导致定义在Student类中的toString方法被调用。

调用m(new Person())（第5行）会导致定义在Person类中的toString方法被调用。调用m(new Object())（第6行）会导致定义在Object类中的toString方法被调用。

匹配方法的签名和绑定方法的实现是两个独立的事情。引用变量的*声明类型*决定了编译时匹配哪个方法。编译器会在编译时，根据参数类型、参数个数和参数顺序找到匹配的方法。一个方法可能在几个子类中都被实现。Java虚拟机在运行时动态绑定方法的实现，这是由变量的实际类型决定的。

11.9 对象转换和instanceof运算符

你已经使用过转换运算符把一种基本类型变量转换为另一种基本类型。转换也可以用来把一种类类型的对象转换为继承体系结构中的另一种类类型的对象。在上一节中，语句

```
m(new Student());
```

将对象new Student()赋值给一个Object类型的参数。这条语句等价于

```
Object o = new Student(); // Implicit casting
m(o);
```

由于Student的实例自动地就是Object的实例，所以，语句Object o = new Student()是合法的，它称为隐式转换（implicit casting）。

假设想使用下面的语句把对象引用o赋值给Student类型的变量：

```
Student b = o;
```

在这种情况下，将会发生编译错误。为什么语句Object o = new Student()可以运行，而语句Student b = o不行呢？原因是Student对象总是Object的实例，但是，Object对象不一定是Student的实例。即使可以看到o实际上是一个Student的对象，但是编译器还没有聪明到懂得这一点。为了告诉编译器o就是一个Student对象，就要使用显式转换（explicit casting）。它的语法与基本类型转换的语法很类似，用圆括弧把目标对象的类型括住，然后放到要转换的对象前面，如下所示：

```
Student b = (Student)o; // Explicit casting
```

总是可以将一个子类的实例转换为一个父类的变量（称之为向上转换upcasting），因为子类的实例永远是它的父类的实例。当把一个父类的实例转换为它的子类变量（称之为向下转换downcasting）时，必须使用转换记号“(子类名)”进行显式转换，向编译器表明你的意图。为使转换成功，必须确保要转换的对象是子类的一个实例。如果父类对象不是子类的一个实例，就会出现一个运行异常ClassCastException。例如：如果一个对象不是Student的实例，它就不能转换成Student类型的变量。因此，一个好的经验是，在尝试转换之前，确保该对象是另一个对象的实例。这是可以利用运算符instanceof来实现的。考虑下面的代码：

```
Object myObject = new Circle();
... // Some lines of code

/** Perform casting if myObject is an instance of Circle */
if (myObject instanceof Circle) {
  System.out.println("The circle diameter is " +
    ((Circle)myObject).getDiameter());
  ...
}
```

你可能会奇怪为什么必须进行类型转换。变量myObject被声明为Object。声明类型决定了在编译时匹配哪个方法。使用myObject.getDiameter()会引起一个编译错误，因为Object类没有getDiameter方法。编译器无法找到和myObject.getDiameter()匹配的方法。所以，有必要将myObject转换成Circle类型，来告诉编译器myObject也是Circle的一个实例。

为什么没有在一开始就把myObject定义为Circle类型呢？为了能够进行通用程序设计，一个好的经验是把变量定义为父类型，这样，它就可以接收任何子类型的值。

注意 instanceof是Java的关键字。在Java关键字中的每个字母都是小写的。

提示 为了更好地理解类型转换，可以认为它们类似于水果、苹果、橘子之间的关系，其中水果类Fruit是苹果类Apple和橘子类Orange的父类。苹果是水果，所以，总是可以将Apple的实例安全地赋值给Fruit变量。但是，水果不一定是苹果，所以，必须进行显式转换才能将Fruit的实例赋值给Apple的变量。

程序清单11-7演示了多态和类型转换。程序创建两个对象（第5～6行），一个圆和一个矩形，然后调用displayObject方法显示它们（第9～10行）。如果对象是一个圆，displayObject方法显示它的面积和周长（第15行），而如果对象是一个矩形，这个方法显示它的面积（第21行）。

程序清单11-7 CastingDemo.java

```
1 public class CastingDemo {
2   /** Main method */
3   public static void main(String[] args) {
4     // Create and initialize two objects
5     Object object1 = new Circle4(1);
6     Object object2 = new Rectangle1(1, 1);
```

```
 7
 8     // Display circle and rectangle
 9     displayObject(object1);
10     displayObject(object2);
11   }
12
13   /** A method for displaying an object */
14   public static void displayObject(Object object) {
15     if (object instanceof Circle4) {
16       System.out.println("The circle area is " +
17         ((Circle4)object).getArea());
18       System.out.println("The circle diameter is " +
19         ((Circle4)object).getDiameter());
20     }
21     else if (object instanceof Rectangle1) {
22       System.out.println("The rectangle area is " +
23         ((Rectangle1)object).getArea());
24     }
25   }
26 }
```

```
The circle area is 3.141592653589793
The circle diameter is 2.0
The rectangle area is 1.0
```

displayObject(Object object)方法是一个通用程序设计的例子。它可以通过传入Object的任何实例被调用。

程序使用隐式转换将一个Circle对象赋值给object1并且将一个Rectangle对象赋值给object2（第5～6行），然后调用displayObject方法显示这些对象的信息（第9～10行）。

在displayObject方法中（第14～25行），如果对象是Circle的一个实例，则用显式转换将这个对象转换为Circle对象。使用getArea和getDiameter方法显示这个圆的面积和直径。

只有源对象是目标类的实例时才能进行类型转换。在执行转换前，程序使用instanceof运算符来确保源对象是否是目标类的实例（第15行）。

由于getArea和getDiameter方法在Object类中是不可用的，所以，有必要显式地转换成Circle类型（第17、19行）和Rectangle类型（第23行）。

警告 对象成员访问运算符（.）优先于类型转换运算符。使用圆括号保证在点运算符（.）之前进行转换，例如：

```
((Circle)object).getArea());
```

11.10 Object的equals方法

在Object中定义的另外一个经常使用的方法是equals方法。它的签名是：

```
public boolean equals(Object o)
```

这个方法测试两个对象是否相等。调用equals的语法是：

```
object1.equals(object2);
```

Object类中equals方法的默认实现是：

```
public boolean equals(Object obj) {
  return (this == obj);
}
```

这个实现使用==运算符检测两个引用变量是否指向同一个对象。因此，应该在自己的客户类中覆盖这个方法，以测试两个不同的对象是否具有相同的内容。

在9.2节中已经用过equals方法比较两个字符串。String类中的equals方法继承自Object类，然后在String类中被覆盖，使之能够检验两个字符串的内容是否相等。可以覆盖Circle类中的equals方法，基于圆的半径比较两个圆是否相等，如下所示：

```
public boolean equals(Object o) {
  if (o instanceof Circle) {
    return radius == ((Circle)o).radius;
  }

  else
    return false;
}
```

注意 比较运算符==用来比较两个基本数据类型的值是否相等，或者判断两个对象是否具有相同的引用。如果想让equals方法能够判断两个对象是否具有相同的内容，可以在定义这些对象的类时，覆盖Circle类中的equals方法。运算符==要比equals方法的功能强大些，因为==运算符可以检测两个引用变量是否指向同一个对象。

警告 在子类中，使用签名equals(SomeClassName obj)（例如：equals(Circle c)）覆盖equals方法是一个常见错误。应该使用equals(Object obj)。参见复习题11.15。

11.11 数组线性表ArrayList类

现在，我们准备介绍一个很有用的存储对象的类。可以创建一个数组存储对象。但是，这个数组一旦创建，它的大小就固定了。Java提供ArrayList类来存储不限定个数的对象。图11-3给出ArrayList中的一些方法。

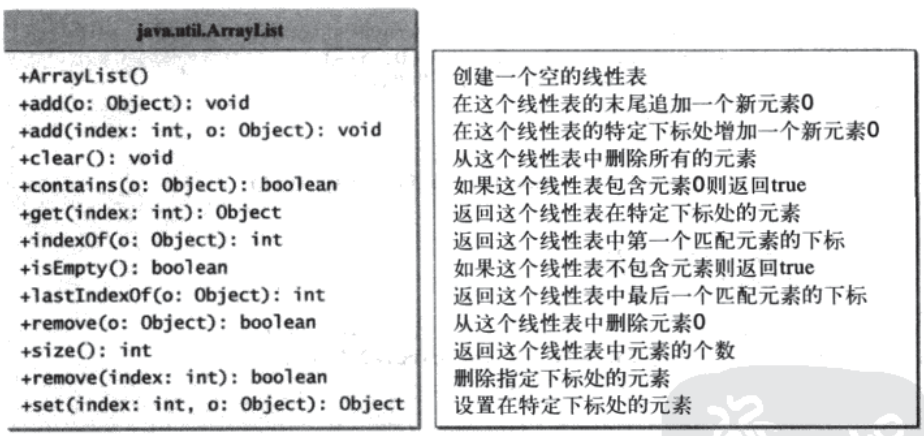


图11-3 ArrayList中存储不限定个数的对象

程序清单11-8是使用ArrayList存储对象的一个例子。

程序清单11-8 TestArrayList.java

```
1  public class TestArrayList {
2    public static void main(String[] args) {
3      // Create a list to store cities
4      java.util.ArrayList cityList = new java.util.ArrayList();
5
6      // Add some cities in the list
7      cityList.add("London");
8      // cityList now contains [London]
9      cityList.add("Denver");
10     // cityList now contains [London, Denver]
```

```
11      cityList.add("Paris");
12      // cityList now contains [London, Denver, Paris]
13      cityList.add("Miami");
14      // cityList now contains [London, Denver, Paris, Miami]
15      cityList.add("Seoul");
16      // contains [London, Denver, Paris, Miami, Seoul]
17      cityList.add("Tokyo");
18      // contains [London, Denver, Paris, Miami, Seoul, Tokyo]
19
20      System.out.println("List size? " + cityList.size() );
21      System.out.println("Is Miami in the list? " +
22        cityList.contains("Miami") );
23      System.out.println("The location of Denver in the list? "
24        + cityList.indexOf("Denver"));
25      System.out.println("Is the list empty? " +
26        cityList.isEmpty() ); // Print false
27
28      // Insert a new city at index 2
29      cityList.add(2, "Xian");
30      // contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
31
32      // Remove a city from the list
33      cityList.remove("Miami");
34      // contains [London, Denver, Xian, Paris, Seoul, Tokyo]
35
36      // Remove a city at index 1
37      cityList.remove(1);
38      // contains [London, Xian, Paris, Seoul, Tokyo]
39
40      // Display the contents in the list
41      System.out.println(cityList.toString());
42
43      // Display the contents in the list in reverse order
44      for (int i = cityList.size() - 1; i >= 0; i--)
45        System.out.print(cityList.get(i) + " ");
46      System.out.println();
47
48      // Create a list to store two circles
49      java.util.ArrayList list = new java.util.ArrayList();
50
51      // Add two circles
52      list.add(new Circle4(2));
53      list.add(new Circle4(3));
54
55      // Display the area of the first circle in the list
56      System.out.println("The area of the circle? " +
57        ((Circle4)list.get(0)).getArea());
58    }
59 }
```

```
List size? 6
Is Miami in the list? true
The location of Denver in the list? 1
Is the list empty? false
[London, Xian, Paris, Seoul, Tokyo]
Tokyo Seoul Paris Xian London
The area of the circle? 12.566370614359172
```

程序使用无参构造方法创建一个ArrayList（第4行），add方法将Object的任一实例加入线性表中。由于String是Object的一个子类，所以字符串可以加入到线性表中。add方法（第7～17行）将一个对象加入到线性表的末尾。所以，在执行完cityList.add("London")（第7行）之后，这个线性表包含

```
[London]
```

执行完cityList.add("New Denver")（第9行）后，这个线性表包含

```
[London, Denver]
```

在加入Paris、Miami、Seoul和Tokyo之后（第11～17行），这个线性表包含

```
[London,Denver,Paris,Miami,Seoul,Tokyo]
```

调用size()（第20行）返回这个线性表的大小，线性表的当前大小为6。调用contains ("Miami")（第22行）检测这个对象是否在这个线性表中。在这种情况下，它返回true，因为Miami在这个线性表中。调用indexOf("Denver")（第24行）返回该对象在线性表中的索引值，这里它的值为1。如果对象不在这个线性表中，它返回-1。isEmpty()方法（第26行）检测这个线性表是否为空。因为当前列表不为空，所以它返回false。

语句cityList.add(2,"Xian")（第29行）在这个线性表的指定下标位置插入一个对象。该语句执行完之后，线性表变成

```
[London,Denver,Xian,Paris,Miami,Seoul,Tokyo]
```

语句cityList.remove("Miami")（第33行）从线性表中删除该对象。该语句执行后，线性表就变成

```
[London,Denver,Xian,Paris,Seoul,Tokyo]
```

语句cityList.remove(1)语句（第37行）从线性表中删除指定下标位置的对象，该语句执行后，线性表变成

```
[London,Xian,Paris,Seoul,Tokyo]
```

第41行的语句相当于

```
System.out.println(cityList);
```

方法toString()返回表示线性表的字符串，其形式为[e0.toString(),e1.toString(),…,ek.toString()]，这里的e0，e1，…，ek都是线性表中的元素。…

方法get(index)（第45行）返回指定下标位置处的对象。

注意 从命令行编译这个程序时，会产生下面的警告：

```
Note: TestArrayList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

这个警告可以使用第21章中介绍的泛型类型来消除。目前就先忽略它。尽管会有这个警告，但是程序还是会编译，产生一个.class文件。

可以像使用数组一样使用ArrayList对象，但是两者还是有很多不同之处。表11-1列出了它们的异同点。

一旦创建了一个数组，它的大小就确定下来了。可以使用方括号访问数组元素（例如：a[index]）。当创建ArrayList后，它的大小为0。如果元素不在线性表中，就不能使用get和set方法。向线性表中添加、插入和删除元素是比较容易的，而向数组中添加、插入和删除元素是比较复杂的。为了实现这些操作，必须编写代码操纵这个数组。

表11-1 数组和ArrayList之间的异同

操 作	数 组	ArrayList
创建数组/ArrayList	`Object[] a = new Object[10]`	`ArrayList list = new . ArrayList()`
引用元素	`a [index]`	`list.get(index)`
更新元素	`a [index] = "London";`	`list.set(index, "London");`
返回大小	`a length`	`list.size()`
添加一个新元素		`list.add("London")`
插入一个新元素		`list.add(index, "London")`
删除一个元素		`list.remove(index)`
删除一个元素		`list.remove(Object)`
删除所有元素		`list.clear()`

注意 java.util.Vector也是一个存储对象的类，它与ArrayList类非常相似。Vector也有ArrayList中的所有方法。Vector类是在JDK 1.1中介绍的。在JDK 1.2中介绍ArrayList类是为了代替Vector类。

11.12 自定义栈类

10.9节给出了一个栈类存放int值。本节介绍存储对象的栈类。可以使用ArrayList实现Stack，如程序清单11-9所示。该类的UML图如图11-4所示。

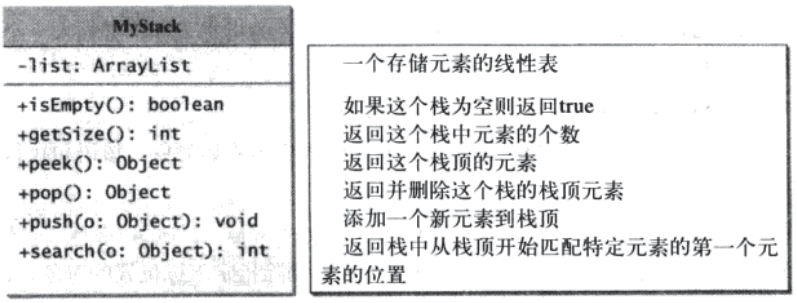


图11-4 MyStack类封装栈的存储并提供处理这个栈的操作

程序清单11-9 MyStack.java

```
1 public class MyStack {
2   private java.util.ArrayList list = new java.util.ArrayList();
3
4   public boolean isEmpty() {
5     return list.isEmpty();
6   }
7
8   public int getSize() {
9     return list.size();
10  }
11
12  public Object peek() {
13    return list.get(getSize() - 1);
14  }
15
16  public Object pop() {
17    Object o = list.get(getSize() - 1);
18    list.remove(getSize() - 1);
19    return o;
20  }
21
22  public void push(Object o) {
23    list.add(o);
24  }
25
26  public int search(Object o) {
27    return list.lastIndexOf(o);
28  }
29
30  /** Override the toString in the Object class */
31  public String toString() {
32    return "stack: " + list.toString();
33  }
34 }
```

创建数组线性表来存储栈中的元素（第2行）。isEmpty()方法（第4～6行）返回list.isEmpty()。

getSize()方法（第8～10行）返回list.size()。peek()方法（第12～14行）可以获取栈顶元素而不删除它，线性表末尾的元素正是栈顶的元素。pop()方法（第16～20行）删除栈顶元素并返回该元素。push(Object element)方法（第22～24行）将指定元素添加到这个栈中。search(Object element)方法检测指定元素是否在栈中，并通过调用list.lastIndexOf(o)返回从栈顶开始第一个匹配元素的索引。通过调用list.toString()覆盖Object类中定义的toString()方法（第31～33行）显示这个栈中的内容。ArrayList中实现的toString()返回表示一个数组线性表中所有元素的字符串表示。

设计指南 在程序清单11-9中，MyStack中包含ArrayList。MyStack和ArrayList之间的关系为组合。因为继承是对“是一种”（is-a）关系建模，组合是对“是一部分”（has-a）关系建模。可以将MyStack实现为ArrayList的一个子类（参见练习题11.4）。使用组合关系更好些，因为它可以定义一个全新的类，而无须继承ArrayList中不必要和不恰当的方法。

11.13 protected数据和方法

目前，已经使用过关键字private和public来表示是否可以从类外访问这些数据域。私有成员只能在类内访问，而公共成员可以被其他类访问。

经常需要允许子类访问定义在父类中的数据域或方法，但不允许非子类访问这些数据域和方法。可以使用关键字protected完成该功能。父类中被保护的数据域或方法可以在它的子类中访问。

修饰符private、protected和public都称为可见性修饰符（visibility modifier）或可访问性修饰符（accessibility modifier），因为它们指定如何访问类和类的成员。这些修饰符的可见性按下面的顺序递增：

可见性增加 →

private、none（如果不使用任何修饰符）、protected、public

表11-2总结了类中成员的可访问性。图11-5描述了C1类中的public、protected、默认的和private数据或方法是如何被C2、C3、C4和C5类访问的，其中，C2类与C1类在同一个包中、C3类是C1类在同一个包中的子类、C4类是C1类在不同包中的子类、C5类与C1类在不同包中。

表11-2 数据和方法的可见性

类中成员的修饰符	在同一类内可访问	在同一包内可访问	在子类内可访问	在不同包可访问
public	✓	✓	✓	✓
protected	✓	✓	✓	–
(default)	✓	✓	–	–
private	✓	–	–	–

使用private修饰符可以完全隐藏类的成员，这样，就不能从类外直接访问它们。不使用修饰符就表示允许同一个包里的任何类直接访问类的成员，但是其他包中的类不可以访问。使用protected修饰符允许任何包中的子类或同一包中的类访问类的成员。使用public修饰符允许任意类访问类的成员。

类可以以两种方式使用：一种是为创建该类的实例；另一种是通过扩展该类创建它的子类。如果不想从类外使用类的成员，就把成员声明成private。如果想让该类的用户都能使用类的成员，就把成员声明成public。如果想让该类的扩展者使用数据和方法，而不想让该类的用户使用，则把成员声明成protected。

修饰符private和protected只能用于类的成员。public修饰符和默认修饰符（也就是没有修饰符）既可以用于类的成员，同样也可以用于类。一个没有修饰符的类（即非公共类）是不能被其他包中的类访问的。

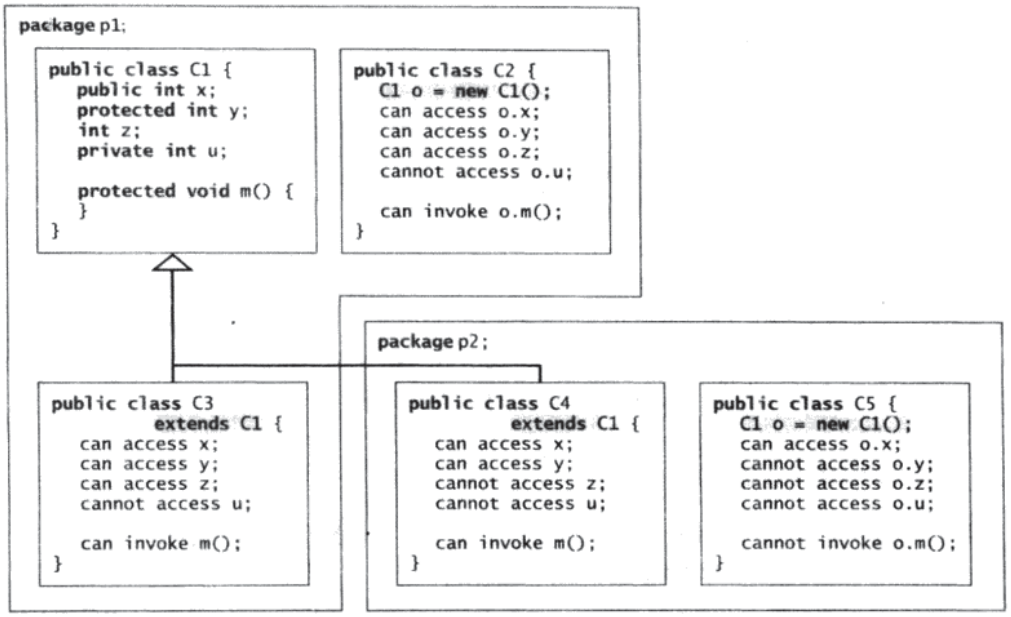


图11-5 使用可见性修饰符控制如何访问数据和方法

注意 子类可以覆盖它的父类的protected方法，并把它的可见性改为public。但是，子类不能削弱父类中定义的方法的可访问性。例如：如果一个方法在父类中定义为public，在子类中也必须定义为public。

11.14 防止扩展和覆盖

有时候，可能希望防止类扩展。在这种情况下，使用final修饰符表明一个类是终极的，是不能作为父类的。Math类就是一个终极类。String、StringBuilder和StringBuffer类也可以是终极类。例如，下面的类就是终极的，是不能扩展的：

```
public final class C {
  // Data fields, constructors, and methods omitted
}
```

也可以定义一个方式为终极的，一个终极方法不能被它的子类覆盖。

例如，下面的方法是终极的，是不能覆盖的：

```
public class Test {
  // Data fields, constructors, and methods omitted

  public final void m() {
    // Do something
  }
}
```

注意 修饰符可以用在类和类的成员（数据和方法）上，只有final修饰符还可以用在方法中的局部变量上。方法内的终极局部变量就是常量。

关键术语

actual type（实际类型）
array list（数组线性表）
casting object（转换对象）
composition（组合）

constructor chaining（构造方法链）
declared type（声明类型）
dynamic binding（动态绑定）
final（终极修饰符）
has-a relationship（是一部分的关系）
inheritance（继承）
instanceof（运算符instanceof）
is-a relationship（是一种的关系）
override（覆盖）
polymorphism（多态）
protected（保护修饰符）
subclass（子类）
subtype（子类型）
superclass（父类）
supertype（父类型）
vector（矢量）

本章小结

- 可以从现有的类派生出新类。这称为类的继承。新类称为次类、子类或派生类。现有的类称为超类、父类或基类。
- 构造方法用来构造类的实例。不同于属性和方法，子类不继承父类的构造方法。它们只能用关键字super从子类的构造方法中调用。
- 构造方法可以调用重载的构造方法或它的父类的构造方法。这种调用必须是构造方法的第一条语句。如果没有显式地调用它们中的任何一个，编译器就会把super()作为构造方法的第一条语句，它调用的是父类的无参构造方法。
- 为了覆盖一个方法，必须使用与它的父类中的方法相同的签名来定义子类中的方法。
- 实例方法只有是可访问的时候才能覆盖。这样，私有方法是不能覆盖的，因为它是不能在类本身之外访问的。如果子类中定义的方法在父类中是私有的，那么这两个方法是完全没有关系的。
- 静态方法与实例方法一样可以继承。但是，静态方法不能覆盖，如果父类中定义的静态方法在子类中重新定义，那么父类中定义的方法被隐藏。
- Java中的每个类都源于java.lang.Object类。如果一个类在定义时没有指定继承关系，那么它的父类就是Object。
- 如果一个方法的参数类型是父类（例如：Object），可以向该方法的参数传递任何子类（例如：Circle类或String类）的对象。当在方法中使用一个对象（例如：Circle对象或String对象）时，动态地决定调用该对象方法（例如：toString）的某个特定的实现。
- 因为子类的实例总是它的父类的实例，所以，总是可以将一个子类的实例转换成一个父类的变量。当把父类实例转换成它的子类变量时，必须使用转换记号（子类名）进行显式转换，向编译器表明你的意图。
- 一个类定义一个类型。子类定义的类型称为子类型，而父类定义的类型称为父类型。
- 当从引用变量调用实例方法时，该变量的实际类型在运行时决定使用该方法的哪个实现。当访问数据域或静态方法时，引用变量的声明类型在编译时决定使用哪个方法。
- 可以使用表达式obj instanceof AClass（对象名instanceof类名）测试一个对象是否是一个类的实例。
- 可以使用protected修饰符来防止方法和数据被不同包的非子类访问。
- 可以用final修饰符来表明一个类是终极的，是不能成为父类的；并且用它来表明一个方法是终极的，是不能覆盖的。

复习题

11.2～11.5节

11.1 运行图a中的类C时，输出结果是什么？图b中的程序在编译时会出现什么问题？

```
class A {
  public A() {
    System.out.println(
      "A's no-arg constructor is invoked");
  }
}

class B extends A {
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}
```

a)

```
class A {
  public A(int x) {
  }
}

class B extends A {
  public B() {
  }
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}
```

b)

11.2 下面的说法是对还是错？

(1) 子类是父类的一个子集。

(2) 当从子类调用一个构造方法时，它的父类的无参构造方法总是被调用。

(3) 可以覆盖定义在父类中的私有方法。

(4) 可以覆盖定义在父类中的静态方法。

11.3 指出下面类中的问题：

```
1 public class Circle {
2   private double radius;
3
4   public Circle(double radius) {
5     radius = radius;
6   }
7
8   public double getRadius() {
9     return radius;
10  }
11
12  public double getArea() {
13    return radius * radius * Math.PI;
14  }
15 }
16
17 class B extends Circle {
18   private double length;
19
20   B(double radius, double length) {
21     Circle(radius);
22     length = length;
23   }
24
25   /** Override getArea() */
26   public double getArea() {
27     return getArea() * length;
28   }
29 }
```

11.4 如何从子类显式调用父类的构造方法？

11.5 如何从子类调用被覆盖的父类方法？

11.6 解释方法覆盖和方法重载之间的区别。

11.7 如果子类中的一个方法具有和它父类中的方法完全相同的方法头，且返回值类型也相同，那么这是方法的覆盖还是重载呢？

11.8 如果子类中的一个方法具有和它父类中的方法完全相同的方法头，但返回值类型不相同，这会是一个问题吗？

11.9 如果子类中的一个方法具有和它父类中的方法相同的名字，但参数类型不同，那么这是方法的覆盖还是重载呢？

11.6～11.9节

11.10 是否每个类都有toString方法和equals方法？它们是从哪儿来的？如何使用它们？覆盖这些方法是否合适？

11.11 给出下面程序的输出：

```
1 public class Test {
2   public static void main(String[] args) {
3     A a = new A(3);
4   }
5 }
6
7 class A extends B {
8   public A(int t) {
9     System.out.println("A's constructor is invoked");
10  }
11 }
12
13 class B {
14   public B() {
15     System.out.println("B's constructor is invoked");
16   }
17 }
```

当调用new A(3)时，会调用Object的无参构造方法吗？

11.12 对于程序清单11-1和程序清单11-2中的GeometricObject类和Circle类，回答下面的问题：

(1) 下面的布尔表达式的值是true还是false？

```
Circle circle = new Circle(1);
GeometricObject object1 = new GeometricObject();
(circle instanceof GeometricObject)
(object1 instanceof GeometricObject)
(circle instanceof Circle)
(object1 instanceof Circle)
```

(2) 下面的语句能够编译吗？

```
Circle circle = new Circle(5);
GeometricObject object = circle;
```

(3) 下面的语句能够编译吗？

```
GeometricObject object = new GeometricObject();
Circle circle = (Circle)object;
```

11.13 假设Fruit、Apple、Orange、GoldenDelicious和Macintosh声明为如图11-6所示。

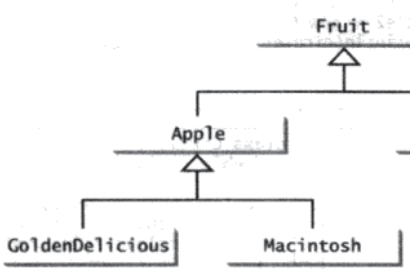


图11-6 GoldenDelicious和Macintosh是Apple的子类；Apple和Orange是Fruit的子类

假设给出下面的声明：

```
Fruit fruit = new GoldenDelicious();
Orange orange = new Orange();
```

回答下面的问题：

(1) fruit instanceof Fruit的值为true吗？

(2) fruit instanceof Orange的值为true吗?

(3) fruit instanceof Apple的值为true吗?

(4) fruit instanceof GoldenDelicious的值为true吗?

(5) fruit instanceof Macintosh的值为true吗?

(6) orange instanceof Orange的值为true吗?

(7) orange instanceof Fruit的值为true吗?

(8) orange instanceof Apple的值为true吗?

(9) 假设makeApple Cider方法定义在Apple类中。fruit可以调用这个方法吗? orange可以调用这个方法吗?

(10) 假设makeOrangeJuice方法定义在Orange类中。orange可以调用这个方法吗? fruit可以调用这个方法吗?

(11) 语句Orange p=new Apple()是否合法?

(12) 语句Macintosh p=new Apple()是否合法?

(13) 语句Apple p=new Macintosh()是否合法?

11.14 下面代码中的错误是什么?

```
1 public class Test {
2   public static void main(String[] args) {
3     Object fruit = new Fruit();
4     Object apple = (Apple)fruit;
5   }
6 }
7
8 class Apple extends Fruit {
9 }
10
11 class Fruit {
12 }
```

11.10节

11.15 当覆盖equals方法时，常见的错误就是在子类中输错它的签名。例如：equals方法被错误地写成equals(Circle circle)，如下面图a中的代码所示；应该使用如图b中所示的equals(Object circle)替换它。分别给出运行图a和图b中的Test类和Circle类的输出。

```
public class Test {
  public static void main(String[] args) {
    Object circle1 = new Circle();
    Object circle2 = new Circle();
    System.out.println(circle1.equals(circle2));
  }
}
```

```
class Circle {
  double radius;

  public boolean equals(Circle circle) {
    return this.radius == circle.radius;
  }
}
```

a)

```
class Circle {
  double radius;

  public boolean equals(Object circle) {
    return this.radius ==
      ((Circle)circle).radius;
  }
}
```

b)

11.11～11.12节

11.16 如何创建一个ArrayList? 如何向线性表中追加一个对象? 如何在线性表的开始位置插入一个对象? 如何找出线性表中所包含对象的个数? 如何从线性表中删除给定对象? 如何从线性表中删除最后的对象? 如何检测一个给定的对象是否在线性表中? 如何从线性表中获取指定下标位置的对象?

11.17 下面代码中有三处错误，请找出它们:

```
ArrayList list = new ArrayList();
list.add("Denver");
list.add("Austin");
list.add(new java.util.Date());
String city = list.get(0);
list.set(3, "Dallas");
System.out.println(list.get(3));
```

11.13～11.14节

11.18 应该在类上使用什么修饰符，才能使同一个包中的类可以访问它，而不同包中的类不能访问它？

11.19 应该用什么修饰符，才能使不同包中的类不能访问这个类，而任何包中的子类都可以访问它？

11.20 在下面的代码中，类A和类B在同一个包中。如果问号被空白代替，那么类B能编译吗？如果问号被private代替，那么类B能编译吗？如果问号被protected代替，类B能编译吗？

```
package p1;

public class A {
  ?   int i;
  ?   void m() {
    ...
  }
}
```

a)

```
package p1;

public class B extends A {
  public void m1(String[] args) {
    System.out.println(i);
    m();
  }
}
```

b)

11.21 在下面的代码中，类A和类B在不同的包中。如果问号被空白代替，那么类B能编译吗？如果问号被private代替，那么类B能编译吗？如果问号被protected代替，那么类B能编译吗？

```
package p1;

public class A {
  ?   int i;
  ?   void m() {
    ...
  }
}
```

a)

```
package p2;

public class B extends A {
  public void m1(String[] args) {
    System.out.println(i);
    m();
  }
}
```

b)

11.22 如何防止一个类被扩展？如何防止一个方法被覆盖？

综合题

11.23 定义下列术语：继承、父类、子类、关键字super和this、转换对象、修饰符protected和final。

11.24 确定下面语句是对还是错：

(1) 被保护的数据或方法可以被同一包中的任何类访问。

(2) 被保护的数据或方法可以被不同包中的任何类访问。

(3) 被保护的数据或方法可以被任意包中它的子类访问。

(4) 终极类可以有实例。

(5) 终极类可以被扩展。

(6) 终极方法可以被覆盖。

(7) 总可以成功地将子类的实例转换为父类。

(8) 总可以成功地将父类的实例转换为子类。

11.25 描述方法匹配和方法绑定之间的区别。

11.26 什么是多态？什么是动态绑定？

编程练习题

11.2～11.4节

11.1 （三角形类Triangle）设计一个名为Triangle的类来扩展GeometricObject类。该类包括：

（1）三个名为side1、side2和side3的double数据域表示这个三角形的三条边，它们的默认值是1.0。

（2）一个无参构造方法创建默认的三角形。

（3）一个能创建带指定side1、side2和side3的三角形的构造方法。

（4）所有三个数据域的访问器方法。

（5）一个名为getArea()的方法返回这个三角形的面积。

（6）一个名为getPerimeter()的方法返回这个三角形的周长。

（7）一个名为toString()的方法返回这个三角形的字符串描述。

计算三角形面积的公式参见练习题2.21。toString()方法的实现如下所示：

```
return "Triangle: side1 = " + side1 + " side2 = " + side2 +
  " side3 = " + side3;
```

画出Triangle类和GeometricObject类的UML图。实现这些类。编写一个测试程序，创建边长为1、1.5和1，颜色为yellow，filled为true的Triangle对象，然后显示它的面积、周长、颜色以及是否被填充。

11.5～11.11节

11.2 （Person、Student、Employee、Faculty和Staff类）设计一个名为Person的类和它的两个名为Student和Employee子类。Employee类又有子类：教员类Faculty和职员类Staff。每个人都有姓名、地址、电话号码和电子邮件地址。学生有班级状态（大一、大二、大三或大四）。将这些状态定义为常量。一个雇员有办公室、工资和受聘日期。定义一个名为MyDate的类，包含数据域：year（年）、month（月）和day（日）。教员有办公时间和级别。职员有职务称号。覆盖每个类中的toString方法，显示相应的类名和人名。

画出这些类的UML图。实现这些类。编写一个测试程序，创建Person、Student、Employee、Faculty和Staff，并且调用它们的toString()方法。

11.3 （账户类Account的子类）在练习题8.7中，定义了一个Account类来建模一个银行账户。一个账户有账号、余额、年利率、开户日期等属性，以及存款和取款等方法。创建两个检测支票账户（checking account）和储蓄账户（saving account）的子类。支票账户有一个透支限定额，但储蓄账户不能透支。

画出这些类的UML图。实现这些类。编写一个测试程序，创建Account、SavingsAccount和CheckingAccount的对象，然后调用它们的toString()方法。

11.4 （利用继承实现MyStack）在程序清单11-9中，MyStack是用组合实现的。扩展ArrayList创建一个新的栈类。

画出这些类的UML图。实现MyStack类。编写一个测试程序，提示用户输入五个字符串，然后以逆序显示这些字符串。

11.5 （课程类Course）改写程序清单10-6中的Course类。使用ArrayList代替数组来存储学生。不应该改变Course类的原始合约（即构造方法和方法的定义都不应该改变）。

11.6 （使用ArrayList）编写程序，创建一个ArrayList，然后向这个线性表中添加一个Loan对象、一个Date对象、一个字符串、一个JFrame对象和一个Circle对象，然后使用循环调用对象的toString()方法，来显示线性表中所有的元素。

***11.7 （实现ArrayList）在Java API中实现ArrayList。实现ArrayList以及图11-3中定义的方法。

提示 使用一个数组存储ArrayList中的元素。如果ArrayList的大小超过当前数组的容量，那就创建一个大小是当前数组两倍的新数组，然后将当前数组的内容复制给新数组。

**11.8 （新的Account类）练习题8.7中给出一个Account类。如下设计一个新的Account类：

- 添加一个String类型的新数据域name来存储客户的名字。
- 添加一个新的构造方法，该方法创建一个带指定名字、id和收支额的账户。
- 添加一个名为transactions的新数据域，它的类型是ArrayList，可以为账户存储交易。每笔交易都是一个Transaction类的实例。Transaction类的定义如图11-7所示。
- 修改withdraw和deposit方法，向transactions数组线性表添加一笔交易。
- 其他所有属性和方法都和练习题8.7中的一样。

编写一个测试程序，创建一个年利率为1.5%、收支额为1000、id为1122而名字为George的Account。向该账户存入30美元、40美元和50美元并从该账户中取出5美元、4美元和2美元。打印账户清单，显示账户持有者名字、利率、收支额和所有的交易。

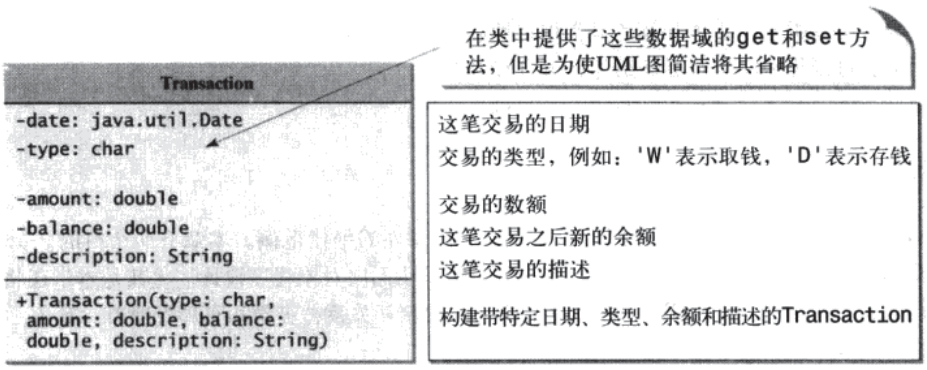


图11-7 Transaction类描述银行账户的一笔交易