第 8 章

接口、继承与多态

（视频讲解：18 分钟）

继承和多态是面向对象开发语言中非常重要的一个环节，如果在程序中使用得当，可以将整个程序的架构变得非常有弹性，同时可以减少代码的冗余性。继承机制的使用可以复用一些定义好的类，减少重复代码的编写。多态机制的使用可以动态调整对象的调用，降低对象之间的依存关系。同时为了优化继承与多态，一些类除了继承父类还使用接口的形式。Java 语言中的类可以同时实现多个接口，接口被用来建立类与类之间关联的标准。在 Java 中正因为使用这些机制使 Java 语言更具有生命力。

通过阅读本章，您可以：

- 掌握接口的使用
- 掌握类的继承
- 掌握 super 关键字的使用方法
- 了解什么是多态

8.1 接口的使用

视频讲解：光盘\TM\lx\8\接口的使用.exe

Java 语言只支持单重继承，不支持多继承，即一个类只能有一个父类。但是在实际应用中，又经常需要使用多继承来解决问题。为了解决该问题，Java 语言提供了接口来实现类的多重继承功能。

8.1.1 接口的定义

使用 interface 来定义一个接口。接口定义与类的定义类似，也是分为接口的声明和接口体，其中接口体由变量定义和方法定义两部分组成。定义接口的基本语法格式如下：

```
[修饰符] interface 接口名 [extends 父接口名列表]{
    [public] [static] [final] 变量;
    [public] [abstract] 方法;
}
```

定义接口的语法格式的参数说明如表 8.1 所示。

表 8.1 定义接口的语法格式的参数说明

参 数	说 明
修饰符	可选参数，用于指定接口的访问权限，可选值为 public。如果省略则使用默认的访问权限
接口名	必选参数，用于指定接口的名称，接口名必须是合法的 Java 标识符。一般情况下，要求首字母大写
extends 父接口名列表	可选参数，用于指定要定义的接口继承于哪个父接口。当使用 extends 关键字时，父接口名为必选参数
方法	接口中的方法只有定义而没有被实现

【例 8.1】 定义一个用于计算的接口，在该接口中定义一个常量 PI 和两个方法。

```
public interface ICalculate {
    final float PI=3.14159f;                    //定义用于表示圆周率的常量 PI
    float getArea(float r);                     //定义一个用于计算面积的方法 getArea()
    float getCircumference(float r);            //定义一个用于计算周长的方法 getCircumference()
}
```

注意

Java 的类文件一样，接口文件的文件名必须与接口名相同。

8.1.2 接口的实现

接口在定义后，就可以在类中实现该接口。在类中实现接口可以使用 implements 关键字，基本语

法格式如下：

```
[修饰符] class <类名> [extends  父类名] [implements  接口列表] {
}
```

实现接口的语法格式的参数说明如表 8.2 所示。

表 8.2　实现接口的语法格式的参数说明

参　数	说　明
修饰符	可选参数，用于指定类的访问权限，可选值为 public、abstract 和 final
类名	必选参数，用于指定类的名称，类名必须是合法的 Java 标识符。一般情况下，要求首字母大写
extends 父类名	可选参数，用于指定要定义的类继承于哪个父类。当使用 extends 关键字时，父类名为必选参数
implements 接口列表	可选参数，用于指定该类实现的是哪些接口。当使用 implements 关键字时，接口列表为必选参数。当接口列表中存在多个接口名时，各个接口名之间使用逗号分隔

在类中实现接口时，方法名、返回值类型、参数的个数及类型必须与接口中的完全一致，并且必须实现接口中的所有方法。

【例 8.2】　编写一个名称为 Cire 的类，该类实现例 8.1 中定义的接口 ICalculate。

```
public class Cire implements ICalculate {
    //定义计算圆面积的方法
    public float getArea(float r) {
        float area=PI*r*r;                          //计算圆面积并赋值给变量 area
        return area;                                //返回计算后的圆面积
    }
    //定义计算圆周长的方法
    public float getCircumference(float r) {
        float circumference=2*PI*r;                 //计算圆周长并赋值给变量 circumference
        return circumference;                       //返回计算后的圆周长
    }
}
```

在类的继承中，只能做单重继承，而实现接口时，一次则可以实现多个接口，每个接口间使用逗号“,”分隔，这时就可能出现变量或方法名冲突的情况。解决该问题时，如果变量冲突，则需要明确指定变量的接口，可以通过“接口名.变量”实现。如果出现方法冲突，则只要实现一个方法即可。下面通过一个具体的实例详细介绍以上问题的解决方法。

8.1.3　范例 1：图片的不同格式保存

在使用图像处理软件处理图片后，需要选择一种格式进行保存。本范例运用接口对图片用不同格式进行保存。运行结果如图 8.1 所示。（**实例位置：光盘\TM\sl\8\1**）

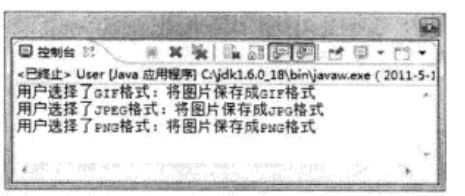

图 8.1　图片的不同格式保存

（1）编写 ImageSaver 接口，在该接口中定义 save()方法。代码如下：

```
public interface ImageSaver {
    void save();                                    //定义 save()方法
}
```

（2）在项目中创建 GIFSaver 类，该类实现了 ImageSaver 接口，在实现 save()方法时将图片保存为 GIF 格式。代码如下：

```
public class GIFSaver implements ImageSaver {
    @Override
    public void save() {                            //实现 save()方法
        System.out.println("将图片保存成 GIF 格式");
    }
}
```

8.1.4 范例 2：为汽车增加 GPS 定位功能

对于刚从工厂生产出来的商品，有些功能并不能完全满足用户的需要，因此，用户通常会对其进行一定的改装工作。本范例将为普通的汽车增加 GPS 定位功能。运行结果如图 8.2 所示。（**实例位置：光盘\TM\ sl\8\2**）

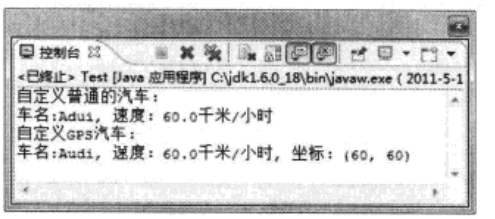

图 8.2 为汽车增加 GPS 定位功能

（1）在项目中创建 Car 类，在该类中首先定义两个属性，一个是 name（表示汽车的名字），另一个是 speed（表示汽车的速度），并为其提供了 getXXX()和 setXXX()方法，然后通过重写 toString()方法来方便输出 Car 对象。代码如下：

```
public class Car {
    private String name;                            //表示汽车的名称
    private double speed;                           //表示汽车的速度
    //省略 getXXX()和 setXXX()方法
    @Override
    public String toString() {                      //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("车名：" + name + ", ");
        sb.append("速度：" + speed + "千米/小时");
        return sb.toString();
    }
}
```

（2）在项目中编写接口 GPS，在该接口中定义了 getLocation()方法，用来确定汽车的位置。代码如下：

```
public interface GPS {
    Point getLocation();                                    //提供定位功能
}
```

（3）在项目中编写 GPSCar 类，该类继承 Car 并实现 GPS 接口。在该类中首先实现 getLocation()方法，用于实现确定汽车位置的功能，然后重写 toString()方法方便输出 GPSCar 对象。代码如下：

```
public class GPSCar extends Car implements GPS {
    @Override
    public Point getLocation() {                                    //利用汽车的速度来确定汽车的位置
        Point point = new Point();
        point.setLocation(super.getSpeed(), super.getSpeed());
        return point;
    }
    @Override
    public String toString() {                                      //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append(", 坐标：(" + getLocation().x + ", " + getLocation().y + ")");
        return sb.toString();
    }
}
```

8.2　类 的 继 承

视频讲解：光盘\TM\lx\8\类的继承.exe

继承一般是指晚辈从父辈那里继承财产，也可以说是子女拥有父母所给予他们的东西。在面向对象程序设计中，继承的含义与此类似，所不同的是，这里继承的实体是类而非人。也就是说继承是子类拥有父类的成员。下面将介绍在 Java 语言中，如何实现类的继承。

8.2.1　继承的实现

在 Java 语言中，继承通过 extends 关键字来实现。也就是用 extends 指明当前类是子类，并指明从哪个类继承而来。即在子类的声明中，通过使用 extends 关键字来显式地指明其父类。其基本的声明格式如下：

```
[修饰符] class 子类名 extends 父类名{
    类体
}
```

- ☑ 修饰符：可选参数，用于指定类的访问权限，可选值为 public、abstract 和 final。
- ☑ 子类名：必选参数，用于指定子类的名称，类名必须是合法的 Java 标识符。一般情况下，要求首字母大写。
- ☑ extends 父类名：必选参数，用于指定要定义的子类继承于哪个父类。

【例 8.3】 定义一个 Pigeon 类，该类继承于父类 Bird，即 Pigeon 类是 Bird 类的子类。（**实例位置：光盘\TM\sl\8\3**）

父类 Bird 的代码如下：

```
public class Bird {
    String color="白色";                    //颜色
    String skin="羽毛";                     //皮毛
}
```

子类 Pigeon 的代码如下：

```
public class Pigeon extends Bird {
    public static void main(String[] args) {
        Pigeon pigeon=new Pigeon();
        System.out.println(pigeon.color);          //输出成员变量 color
    }
}
```

运行结果如图 8.3 所示。

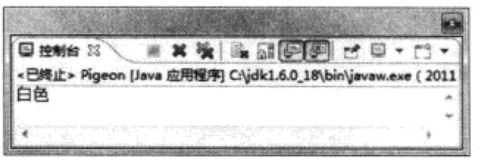

图 8.3 在子类中输出父类的成员变量

8.2.2 继承中的重写

重写是指父子类之间的关系，当子类继承父类中所有可能被子类访问的成员方法时，如果子类的方法名与父类的方法名相同，那么子类就不能继承父类的方法，此时，称为子类的方法重写了父类的方法。重写体现了子类补充或者改变父类方法的能力。通过重写，可以使一个方法在不同的子类中表现出不同的行为。

说明

重写也可以称为覆盖。

【例 8.4】 定义一个动物类 Animal 及它的子类，并在其子类中重写父类的相关方法。（**实例位置：光盘\TM\sl\8\4**）

（1）创建一个名称为 Animal 的类，在该类中声明一个成员方法 cry()。代码如下：

```
public class Animal {
    public Animal() {
    }
    public void cry() {
        System.out.println("动物发出叫声！ ");
    }
}
```

（2）创建一个 Animal 类的子类 Dog，在该类中重写了父类的成员方法 cry()。代码如下：

```
public class Dog extends Animal {
    public Dog() {
    }
    public void cry() {
        System.out.println("狗发出“汪汪……”声！");
    }
}
```

（3）创建一个 Animal 类的子类 Cat，在该类中重写了父类的成员方法 cry()。代码如下：

```
public class Cat extends Animal {
    public Cat() {
    }
    public void cry() {
        System.out.println("猫发出“喵喵……”声！");
    }
}
```

（4）创建一个 Animal 类的子类 Sheep，在该类中不定义任何方法。代码如下：

```
public class Sheep extends Animal {
}
```

（5）创建一个名称为 Zoo 的类，在该类的 main()方法中分别创建子类 Dog、Cat 和 Sheep 的对象并为该对象分配内存，然后分别调用各对象的 cry()方法。代码如下：

```
public class Zoo {
    public static void main(String[] args) {
        Dog dog=new Dog();                                    //创建 Dog 类的对象并为其分配内存
        System.out.println("执行 dog.cry();语句时的输出结果：");
        dog.cry();
        Cat cat=new Cat();                                    //创建 Cat 类的对象并为其分配内存
        System.out.println("执行 cat.cry();语句时的输出结果：");
        cat.cry();
        Sheep sheep=new Sheep();                              //创建 Sheep 类的对象并为其分配内存
        System.out.println("执行 sheep.cry();语句时的输出结果：");
        sheep.cry();
    }
}
```

运行结果如图 8.4 所示。

从上面的运行结果中可以看出，由于 Dog 类和 Cat 类都重写了父类的方法 cry()，所以执行的是子类中的 cry()方法，但是 Sheep 类没有重写父类的方法，所以执行的是父类中的 cry()方法。

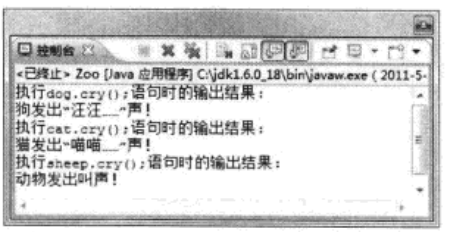

图 8.4　在子类中重写父类的相关方法

8.2.3　使用 super 关键字

子类可以继承父类的非私有成员变量和成员方法（不是以 private 关键字修饰的）作为自己的成员

变量和成员方法。但是，如果子类中声明的成员变量与父类的成员变量同名，则子类不能继承父类的成员变量，此时称子类的成员变量隐藏了父类的成员变量。如果子类中声明的成员方法与父类的成员方法同名，并且方法的返回值及参数个数和类型也相同，则子类不能继承父类的成员方法，此时称子类的成员方法重写了父类的成员方法。这时，如果想在子类中访问父类中被子类隐藏的成员方法或变量，就可以使用 super 关键字。super 关键字主要有以下两种用途。

1．调用父类的构造方法

子类可以调用由父类声明的构造方法。但是必须在子类的构造方法中使用 super 关键字来调用。语法格式如下：

```
super([参数列表]);
```

如果父类的构造方法中包括参数，则参数列表为必选项，用于指定父类构造方法的入口参数。

【例 8.5】 在子类中调用父类的构造方法。（**实例位置：光盘\TM\sl\8\5**）

（1）在项目中创建 Beast 类，在类中添加一个默认的构造方法和一个带参数的构造方法。代码如下：

```
public class Beast {
    String skin = "";                          //成员变量
    public Beast() {                           //默认构造方法
    }
    public Beast(String strSkin) {             //带参数的构造方法
        skin = strSkin;
    }
    public void move() {                       //成员方法
        System.out.println("跑");
    }
}
```

（2）如果想在子类 Tiger 中使用父类的带参数的构造方法，则需要在子类 Tiger 的构造方法中进行调用。代码如下：

```
public class Tiger extends Beast {
    public Tiger (){
        super("条纹");                          //使用父类的带参数的构造方法
    }
}
```

2．操作被隐藏的成员变量和被重写的成员方法

如果想在子类中操作父类中被隐藏的成员变量和被重写的成员方法，也可以使用 super 关键字。语法格式如下：

```
super.成员变量名
super.成员方法名([参数列表])
```

例如，在例 8.5 中，如果想在子类 Tiger 的方法中改变父类 Beast 的成员变量 skin 的值，可以使用

以下代码：

```
super.skin="条纹";
```

如果想在子类 Tiger 的方法中使用父类 Beast 的成员方法 move()，可以使用以下代码：

```
super.move();
```

8.2.4　范例 3：经理与员工的差异

对于在同一家公司工作的经理和员工而言，两者是有很多共同点的。例如每个月都要发工资，但是经理在完成目标任务后，还会获得奖金。此时，利用员工类来编写经理类就会少写很多代码，利用继承技术可以让经理类使用员工类中定义的属性和方法。本范例将通过继承演示经理与员工的差异。运行结果如图 8.5 所示。（**实例位置：光盘\TM\sl\8\6**）

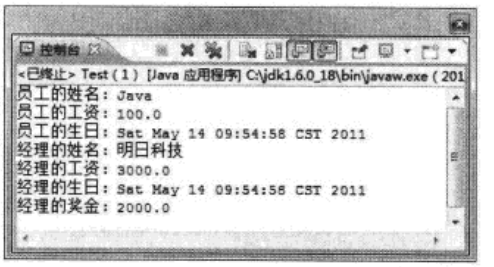

图 8.5　经理与员工的差异

（1）在项目中创建 Employee 类，在该类中定义 3 个属性，分别是 name（表示员工的姓名）、salary（表示员工的工资）和 birthday（表示员工的生日），并分别为它们定义了 getXXX()和 setXXX()方法。代码如下：

```
import java.util.Date;
public class Employee {
    private String name;                          //员工的姓名
    private double salary;                        //员工的工资
    private Date birthday;                        //员工的生日

    public String getName() {                     //获取员工的姓名
        return name;
    }
    public void setName(String name) {            //设置员工的姓名
        this.name = name;
    }

    public double getSalary() {                   //获取员工的工资
        return salary;
    }
    public void setSalary(double salary) {        //设置员工的工资
        this.salary = salary;
    }
    public Date getBirthday() {                   //获取员工的生日
        return birthday;
    }
    public void setBirthday(Date birthday) {      //设置员工的生日
```

```
        this.birthday = birthday;
    }
}
```

（2）在项目中创建一个名称为 Manager 的类，该类继承自 Employee。在该类中，定义了一个 bonus 域，表示经理的奖金，并为其设置了 getXXX()和 setXXX()方法。代码如下：

```
public class Manager extends Employee {
    private double bonus;                          //经理的奖金
    public double getBonus() {                     //获得经理的奖金
        return bonus;
    }
    public void setBonus(double bonus) {           //设置经理的奖金
        this.bonus = bonus;
    }
}
```

（3）在项目中再创建一个名称为 Test 的类，用于测试。在该类中分别创建 Employee 和 Manager 对象，并为其赋值，然后输出其属性。代码如下：

```
import java.util.Date;                             //导入 java.util.Date 类
public class Test {
    public static void main(String[] args) {
        Employee employee = new Employee();        //创建 Employee 对象并为其赋值
        employee.setName("Java");
        employee.setSalary(100);
        employee.setBirthday(new Date());
        Manager manager = new Manager();           //创建 Manager 对象并为其赋值
        manager.setName("明日科技");
        manager.setSalary(3000);
        manager.setBirthday(new Date());
        manager.setBonus(2000);
        //输出经理和员工的属性值
        System.out.println("员工的姓名：" + employee.getName());
        System.out.println("员工的工资：" + employee.getSalary());
        System.out.println("员工的生日：" + employee.getBirthday());
        System.out.println("经理的姓名：" + manager.getName());
        System.out.println("经理的工资：" + manager.getSalary());
        System.out.println("经理的生日：" + manager.getBirthday());
        System.out.println("经理的奖金：" + manager.getBonus());
    }
}
```

8.2.5 范例 4：重写父类中的方法

在继承了一个类之后，就可以使用父类中定义的方法，然而父类中的方法可能并不完全适用于子

类。此时，如果不想定义新的方法，则可以重写父类中的方法。本范例将演示如何重写父类中的方法。运行结果如图 8.6 所示。（**实例位置：光盘\TM\sl\8\7**）

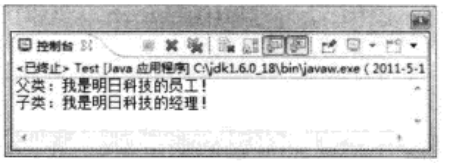

图 8.6　重写父类中的方法

（1）在项目中创建 Employee 类，在该类中添加 getInfo()方法，返回值为字符串“父类：我是明日科技的员工！”。代码如下：

```
public class Employee {
    public String getInfo() {                                    //定义测试用的方法
        return "父类：我是明日科技的员工！";
    }
}
```

（2）在 com.mingrisoft 包中再创建一个名称为 Manager 的类，该类继承自 Employee。在该类中，重写 getInfo()方法。代码如下：

```
public class Manager extends Employee {
    @Override
    public String getInfo() {                                    //重写测试用的方法
        return "子类：我是明日科技的经理！";
    }
}
```

8.3　多　　态

视频讲解：光盘\TM\lx\8\多态.exe

8.3.1　什么是多态

多态性是面向对象程序设计的重要部分。在 Java 语言中，通常使用方法的重载（Overloading）和重写（Overriding）实现类的多态性。其中，重写已经在前面介绍过，下面将对方法的重载进行介绍。

说明　重写之所以具有多态性，是因为父类的方法在子类中被重写，子类和父类的方法名称相同，但完成的功能却不一样，所以说，重写也具有多态性。

方法的重载是指在一个类中出现多个方法名相同，但参数个数或参数类型不同的方法，则称为方法的重载。Java 语言在执行具有重载关系的方法时，将根据调用参数的个数和类型区分具体执行的是哪个方法。下面将通过一个具体的实例进行说明。

【例 8.6】　定义一个名称为 Calculate 的类，在该类中定义两个名称为 getArea()的方法（参数个数不同）和两个名称为 draw()的方法（参数类型不同）。（**实例位置：光盘\TM\sl\8\8**）

```
public class Calculate {
    final float PI=3.14159f;                                    //定义一个用于表示圆周率的常量 PI
    //求圆形的面积
    public float getArea(float r){                              //定义一个用于计算面积的方法 getArea()
        float area=PI*r*r;
        return area;
    }
    //求矩形的面积
    public float getArea(float l,float w){                      //重载 getArea()方法
        float area=l*w;
        return area;
    }
    //画任意形状的图形
    public void draw(int num){                                  //定义一个用于画图的方法 draw()
        System.out.println("画"+num+"个任意形状的图形");
    }
    //画指定形状的图形
    public void draw(String shape){                             //重载 draw()方法
        System.out.println("画一个"+shape);
    }
    public static void main(String[] args) {
        Calculate calculate=new Calculate();                    //创建 Calculate 类的对象并为其分配内存
        float l=20;
        float w=30;
        float areaRectangle=calculate.getArea(l, w);
        System.out.println("求长为"+l+" 宽为"+w+"的矩形的面积是："+areaRectangle);
        float r=7;
        float areaCirc=calculate.getArea(r);
        System.out.println("求半径为"+r+"的圆的面积是："+areaCirc);
        int num=7;
        calculate.draw(num);
        calculate.draw("三角形");
    }
}
```

运行结果如图 8.7 所示。

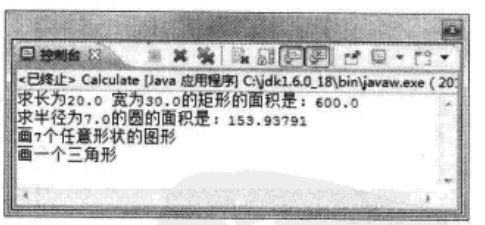

图 8.7　演示方法的重载

说明 重载的方法之间并不一定必须有联系，但是为了提高程序的可读性，一般只重载功能相似的方法。

注意

在进行方法的重载时，方法返回值的类型不能作为区分方法的标志。

8.3.2　范例 5：计算几何图形的面积

对于每个几何图形而言，都有一些共同的属性，如名字和面积等，而其计算面积的方法却各不相同。为了简化开发，本范例将定义一个超类来实现输出名字的方法，并使用抽象方法来计算面积。运行结果如图 8.8 所示。（**实例位置：光盘\TM\sl\8\9**）

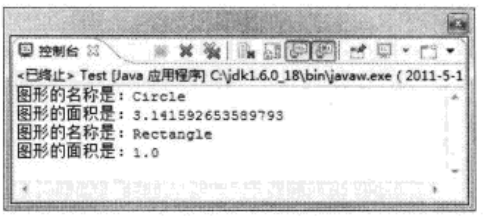

图 8.8　计算几何图形的面积

（1）在项目中创建一个抽象类，名称为 Shape。在该类中定义两个方法，一个是 getName()，用于使用反射机制获得类名称；另一个是抽象方法 getArea()，并未实现。代码如下：

```
public abstract class Shape {
    public String getName() {                          //获得图形的名称
        return this.getClass().getSimpleName();
    }
    public abstract double getArea();                  //获得图形的面积
}
```

（2）在项目中创建一个名称为 Circle 的类，该类继承自 Shape，并实现了抽象方法 getArea()。在该类的构造方法中，获得了圆形的半径，用于在 getArea()方法中计算面积。代码如下：

```
public class Circle extends Shape {
    private double radius;
    public Circle(double radius) {                     //获得圆形的半径
        this.radius = radius;
    }
    @Override
    public double getArea() {                          //计算圆形的面积
        return Math.PI * Math.pow(radius, 2);
    }
}
```

（3）在项目中创建一个名称为 Rectangle 的类，该类继承自 Shape，并实现了抽象方法 getArea()。在该类的构造方法中，获得了矩形的长和宽，用于在 getArea()方法中计算面积。代码如下：

```
public class Rectangle extends Shape {
    private double length;
    private double width;
    public Rectangle(double length, double width) {    //获得矩形的长和宽
```

```
        this.length = length;
        this.width = width;
    }
    @Override
    public double getArea() {                                    //计算矩形的面积
        return length * width;
    }
}
```

（4）在项目中创建一个名称为 Test 的类，用来进行测试，在该类中创建 Circle 和 Rectangle 对象，并分别输出图形的名称和面积。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Circle circle = new Circle(1);                           //创建圆形对象并将半径设置成 1
        System.out.println("图形的名称是：" + circle.getName());
        System.out.println("图形的面积是：" + circle.getArea());
        Rectangle rectangle = new Rectangle(1, 1);               //创建矩形对象并将长和宽设置成 1
        System.out.println("图形的名称是：" + rectangle.getName());
        System.out.println("图形的面积是：" + rectangle.getArea());
    }
}
```

8.3.3 范例 6：简单的汽车销售商场

当顾客在商场购物时，卖家需要根据顾客的需求提取商品。对于汽车销售商场也是如此。用户需要先指定购买的车型，然后商家去提取该车型的汽车。本范例将实现一个简单的汽车销售商场，用来演示多态的用法。运行结果如图 8.9 所示。（**实例位置：光盘\TM\sl\8\10**）

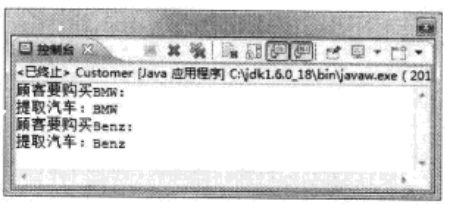

图 8.9 简单的汽车销售商场

（1）在项目中创建一个抽象类，名称为 Car，在该类中定义一个抽象方法 getInfo()。代码如下：

```
public abstract class Car {
    public abstract String getInfo();                            //用来描述汽车的信息
}
```

（2）在项目中创建一个名称为 BMW 的类，该类继承自 Car 并实现了其 getInfo()方法。代码如下：

```
public class BMW extends Car {
    @Override
    public String getInfo() {                                    //用来描述汽车的信息
        return "BMW";
    }
}
```

（3）在项目中创建一个名称为 Benz 的类，该类继承自 Car 并实现了其 getInfo()方法。代码如下：

```
public class Benz extends Car {
    @Override
    public String getInfo() {                                    //用来描述汽车的信息
        return "Benz";
    }
}
```

（4）在项目中创建一个名称为 CarFactory 的类，该类定义了一个静态方法 getCar()，它可以根据用户指定的车型来创建对象。代码如下：

```
public class CarFactory {
    public static Car getCar(String name) {
        if (name.equalsIgnoreCase("BMW")) {                      //如果需要 BMW 则创建 BMW 对象
            return new BMW();
        } else if (name.equalsIgnoreCase("Benz")) {              //如果需要 Benz 则创建 Benz 对象
            return new Benz();
        } else {                                                 //暂时不能支持其他车型
            return null;
        }
    }
}
```

（5）在项目中创建一个名称为 Customer 的类，用来进行测试，在 main()方法中，根据用户的需要提取了不同的汽车。代码如下：

```
public class Customer {
    public static void main(String[] args) {
        System.out.println("顾客要购买 BMW：");
        Car bmw = CarFactory.getCar("BMW");                      //用户要购买 BMW
        System.out.println("提取汽车：" + bmw.getInfo());         //提取 BMW
        System.out.println("顾客要购买 Benz：");
        Car benz = CarFactory.getCar("Benz");                    //用户要购买 Benz
        System.out.println("提取汽车：" + benz.getInfo());        //提取 Benz
    }
}
```

8.4　经 典 范 例

8.4.1　经典范例 1：使用 Comparable 接口自定义排序

视频讲解：光盘\TM\lx\8\使用 Comparable 接口自定义排序.exe

默认情况下，保存在 List 集合中的数组是不进行排序的，但可以通过使用 Comparable 接口自定义

排序规则来自动排序。本范例将介绍如何使用 Comparable 接口自定义排序规则并自动排序。运行结果如图 8.10 所示。（**实例位置：光盘\TM\sl\8\11**）

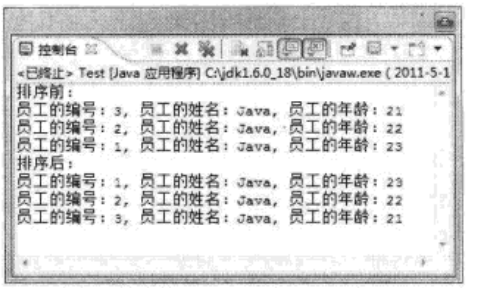

图 8.10　使用 Comparable 接口自定义排序

在项目中创建 Employee 类，在该类中首先定义 3 个属性，分别为 id（表示员工的编号）、name（表示员工的姓名）和 age（表示员工的年龄），然后在构造方法中初始化这 3 个属性，最后再实现接口中定义的 compareTo()方法，将对象按编号升序排列。代码如下：

```
public class Employee implements Comparable<Employee> {
    private int id;                                          //员工的编号
    private String name;                                     //员工的姓名
    private int age;                                         //员工的年龄
    public Employee(int id, String name, int age) {          //利用构造方法初始化各个域
        this.id = id;
        this.name = name;
        this.age = age;
    }
    @Override
    public int compareTo(Employee o) {                       //利用编号实现对象间的比较
        if (id > o.id) {
            return 1;
        } else if (id < o.id) {
            return -1;
        }
        return 0;
    }
    @Override
    public String toString() {                               //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("员工的编号：" + id + ", ");
        sb.append("员工的姓名：" + name + ", ");
        sb.append("员工的年龄：" + age);
        return sb.toString();
    }
}
```

8.4.2　经典范例 2：动态设置类的私有域

视频讲解：光盘\TM\lx\8\动态设置类的私有域.exe

为了保证面向对象的封装特性，通常会将域设置成私有的，然后提供对应的 getXXX()和 setXXX()方法。对于非内部类而言，只能使用 getXXX()和 setXXX()方法来操作该域。然而利用反射机制，就可以在运行时修改类的私有域。本范例将通过简单的 Student 类来演示反射的这种用法。运行结果如图 8.11 所示。（**实例位置：光盘\TM\sl\8\12**）

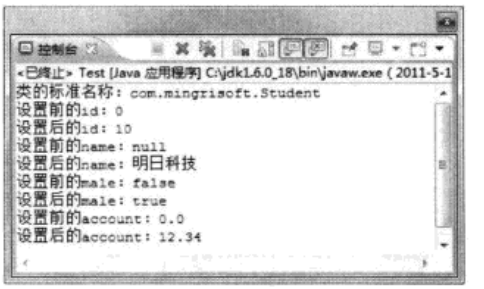

图 8.11　动态设置类的私有域

（1）在项目中创建 Student 类，在该类中定义 4 个域及其对应的 getXXX()和 setXXX()方法。这 4 个域分别是 id（表示学生的序号）、name（表示学生的姓名）、male（表示学生是否为男性）和 account（表示学生的账户余额）。代码如下：

```
public class Student {
    private int id;                                    //表示学生的序号
    private String name;                               //表示学生的姓名
    private boolean male;                              //表示学生的性别
    private double account;                            //表示学生的账户余额
    //省略了各个域的 getXXX()和 setXXX()方法
}
```

（2）在 com.mingrisoft 包中编写 Test 类进行测试。在 main()方法中，分别为不同的域设置不同的值，并输出初始值和新值作为对比。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student();
        Class<?> clazz = student.getClass();                    //获得代表 student 对象的 Class 对象
        System.out.println("类的标准名称：" + clazz.getCanonicalName());
        try {
            Field id = clazz.getDeclaredField("id");
            System.out.println("设置前的 id：" + student.getId());
            id.setAccessible(true);
            id.setInt(student, 10);                             //设置 id 值为 10
            System.out.println("设置后的 id：" + student.getId());

            Field name = clazz.getDeclaredField("name");
            System.out.println("设置前的 name：" + student.getName());
            name.setAccessible(true);
            name.set(student, "明日科技");                       //设置 name 值为明日科技
            System.out.println("设置后的 name：" + student.getName());

            Field male = clazz.getDeclaredField("male");
```

```
            System.out.println("设置前的 male：" + student.isMale());
            male.setAccessible(true);
            male.setBoolean(student, true);                          //设置 male 值为 true
            System.out.println("设置后的 male：" + student.isMale());

            Field account = clazz.getDeclaredField("account");
            System.out.println("设置前的 account：" + student.getAccount());
            account.setAccessible(true);
            account.setDouble(student, 12.34);                       //设置 account 值为 12.34
            System.out.println("设置后的 account：" + student.getAccount());
        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

8.5 本章小结

通过本章的学习，读者可以了解继承与多态的机制，掌握重载、类型转换等技术，学会使用接口与抽象类，从而对继承和多态有一个比较深入的了解。尽管读者已经学习过本章，但笔者还是建议初学者仔细揣摩继承与多态机制，因为继承和多态本身是比较抽象的概念问题，深入理解需要一段时间，使用多态机制必须扩展自己的编程视野，应该将编程的着眼点放在类与类之间的共同特性以及关系上，这样将为软件开发带来更快的速度、更完善的代码组织架构、更好的扩展性和维护性。

8.6 实战练习

1．创建一个抽象类，验证它是否可以实例化对象。（**答案位置：光盘\TM\sl\8\13**）

2．尝试创建一个父类，在父类中创建两个方法，在子类中覆盖第二个方法，为子类创建一个对象，将它向上转型到基类并调用这个方法。（**答案位置：光盘\TM\sl\8\14**）

3．尝试创建一个父类和子类，分别创建构造方法，然后向父类和子类添加成员变量和方法，并总结构建子类对象时的顺序。（**答案位置：光盘\TM\sl\8\15**）