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**）

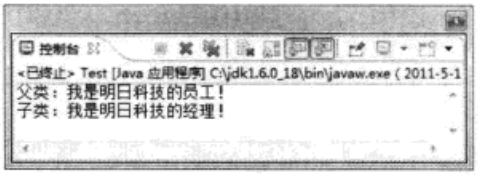


图 8.6　重写父类中的方法

（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。

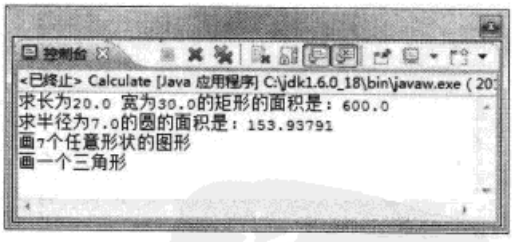


图 8.7　演示方法的重载

**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**）

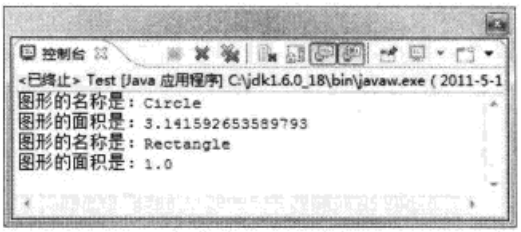


图 8.8　计算几何图形的面积

（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**）

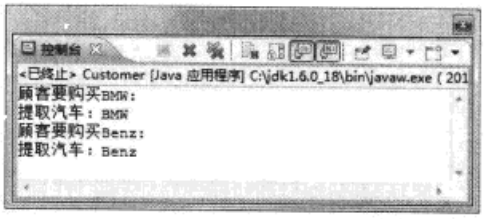


图 8.9 简单的汽车销售商场

（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