

第 5 章

数组及其常用操作

本章读者可以学到如下实例：

- 实例 031 获取一维数组的最小值
- 实例 032 将二维数组中的行列互换
- 实例 033 利用数组随机抽取幸运观众
- 实例 034 用数组设置 JTable 表格的列名与列宽
- 实例 035 使用按钮控件数组实现计算器界面
- 实例 036 通过复选框控件数组实现添加多个复选框控件
- 实例 037 使用选择排序法对数组排序
- 实例 038 使用冒泡排序法对数组排序
- 实例 039 使用快速排序法对数组排序
- 实例 040 使用直接插入法对数组排序
- 实例 041 使用 sort() 方法对数组排序
- 实例 042 反转数组中元素的顺序



实例 031 获得一维数组的最小值

(实例位置: 配套资源\SL\05\031)



实例说明

一维数组常用于保存线性数据,如数据库中的单行数据就可以使用一维数组保存。本实例将接收用户在文本框中输入的单行数据(其中数据都是整数数字,以不同数量的空格分割),这个数据将被程序分解成一维数组,并从数组中提取最小值显示在界面中。实例的运行效果如图 5.1 所示。

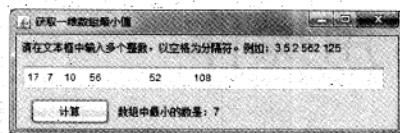


图 5.1 获得一维数组的最小值

指点迷津:

程序经过特殊判断,数字之间的空格可以使用多个。

实现过程

- (1) 在 Eclipse 中创建项目 031, 并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类, 名称为 ArrayMinValue。在窗体中添加一个文本框和一个“计算”按钮以及多个标签控件。
- (3) 编写“计算”按钮的事件处理方法, 在该方法中首先获取用户的输入, 并通过 trim() 方法去除左右空格, 然后对字符串内容进行检测, 排除非法输入, 并把字符串转换为整型数组, 最后在遍历数组的同时提取最小值并显示到窗体的标签控件中。关键代码如下:

```
protected void do_buttonActionPerformed(ActionEvent e) {  
    String arrayStr = textField.getText().trim(); //去除左右空格  
    if(arrayStr.equals("")){ //对空值进行处理  
        JOptionPane.showMessageDialog(null, "请输入数字内容");  
        return;  
    }  
    for (int i = 0; i < arrayStr.length(); i++) { //过滤非法输入  
        char charAt = arrayStr.charAt(i);  
        if (!Character.isDigit(charAt) && charAt != ' ') {  
            JOptionPane.showMessageDialog(null, "输入包含非数字内容");  
            textField.setText("");  
            return;  
        }  
    }  
    String[] numStrs = arrayStr.split(" {1,}"); //分割字符串  
    int[] numArray = new int[numStrs.length]; //创建整型数组  
    //转换输入为整型数组
```

**Note**

```

for (int i = 0; i < numArray.length; i++) {
    numArray[i] = Integer.valueOf(numStrs[i]);
}
int min = numArray[0]; //创建最小数变量
for (int j = 0; j < numArray.length; j++) {
    if (min > numArray[j]) { //提取最小整数
        min = numArray[j];
    }
}
label.setText("数组中最小的数是: " + min); //显示最小值到指定的标签中
}

```

多学两招：

for 语句用于程序的循环流程控制。该语句有 3 个表达式用于循环变量的控制，其完整语法格式为：

```
for(int i=0; i<100; i++){
```

```
    ...
```

```
}
```

for 语句中的 3 个表达式不是完全必备的，可以根据情况部分省略，甚至完全省略。例如下面代码就以最简单的格式实现了无限循环。

```
for (;;) {
```

```
    ...
```

```
}
```

技术要点

本实例主要应用 String 类的 split()方法将输入的字符串分割为字符串数组，再将其转换为整型数组，最后通过 for 循环遍历该数组，并通过 if 语句提取最小值。其中，split()方法可以使字符串按指定的分割字符或字符串进行分割，并将分割后的结果存放在字符串数组中。其语法格式如下：

```
string.split(string sign)
```

参数说明

- string：为字符串对象。
- sign：为分割字符串的分割符，也可以使用正则表达式。

实例 032 将二维数组中的行列互换

(实例位置：配套资源\SL\05\032)

实例说明

数组是程序开发中最常用的，其中二维数组使用最频繁，它可以存储表格数据，还可以根据数组下标索引加入各种运算。另外，图片的关键运算方法也是以二维数组为基础进行矩阵运算的。作为数组知识的巩固，本实例实现数组模拟表格行与列数据交换，这在程序开发中常用。



于表格数据的整理。实例的运行效果如图 5.2 所示。



Note

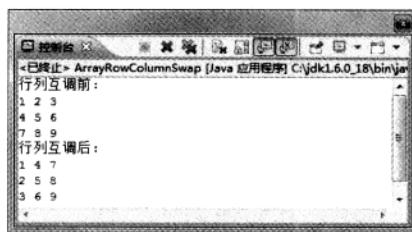


图 5.2 将二维数组中的行列互换

实现过程

(1) 在 Eclipse 中创建项目 032，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 ArrayRowColumnSwap，并在该类的主方法中定义一个二维数组，输出该数组的内容，这次输出是为了与交换数据后的数组进行对比。新创建一个同样大小的二维数组，利用双层 for 循环遍历数组时，把新数组与原数组的行列索引交换进行元素赋值，然后再输出新数组内容。关键代码如下：

```
public class ArrayRowColumnSwap { //创建类
    public static void main(String[] args) {
        //创建二维数组
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        System.out.println("行列互调前: ");
        //输出二维数组
        printArray(arr);
        int arr2[][] = new int[arr.length][arr.length];
        for (int i = 0; i < arr.length; i++) { //调整数组行列数据
            for (int j = 0; j < arr[i].length; j++) {
                arr2[i][j] = arr[j][i];
            }
        }
        System.out.println("行列互调后: ");
        //输出行列互调后的二维数组
        printArray(arr2);
    }
}
```

(3) 编写输出数组内容的 printArray() 方法。输出数组内容的业务在程序中出现两次，根据代码重用的原则，如果相同的业务代码在程序中出现两次以上，就应该把它们提取成一个独立的方法，在这个方法中简单地通过双层 for 循环遍历数组元素。关键代码如下：

```
private static void printArray(int[][] arr) {
    for (int i = 0; i < arr.length; i++) { //遍历数组
        for (int j = 0; j < arr.length; j++) {
            System.out.print(arr[i][j] + " ");
        }
        System.out.println(); //换行
    }
}
```



脚下留神：

在 Java 语言中定义数组变量时，不能声明其长度，只能在使用 new 关键字创建数组时指定，例如 int[9] array =……是错误的写法，应该是 int[] array = new int[9]。



Note

技术要点

本实例应用的主要技术就是创建两个相同大小的二维数组，并使用双层的 for 循环遍历这两个二维数组，同时把新数组与原数组的行列索引交换进行元素赋值，从而实现将二维数组中的行列互换的操作。

实例 033 利用数组随机抽取幸运观众

(实例位置：配套资源\SL\05\033)

实例说明

在电视节目中，经常看到随机抽取幸运观众。如果观众抽取的范围较少，可以让程序使用数组实现，而且效率很高。具体的实现方法是：首先将所有观众姓名生成数组，然后获得数组元素的总数量，最后再随机抽取元素的下标，根据抽取的下标获得幸运观众。本实例将利用数组实现随机抽取幸运观众的程序。实例的运行效果如图 5.3 所示。

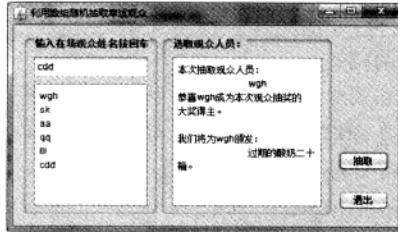


图 5.3 利用数组随机抽取幸运观众

实现过程

- (1) 在 Eclipse 中创建项目 033，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 ArrayExample。在窗体中添加两个文本域、一个文本框和两个按钮，其中两个按钮分别用于抽取幸运观众和退出程序。
- (3) 为文本框添加按键事件监听器，并编写事件处理方法，当用户在文本框中输入观众姓名并按下回车键时，事件处理方法将观众姓名添加到文本域中并以回车换行作为分割符，然后选择文本框中所有文本准备接收用户的下一次输入。关键代码如下：

```
protected void do_TextField_KeyPress(KeyEvent e) {
    if (e.getKeyChar() != '\n') //不是回车字符不做处理
        return;
    String name = nameField.getText();
    if (name.isEmpty()) //如果文本框没有字符串不做处理
        return;
    personnelArea.append(name + "\n"); //把输入人名与回车符添加到人员列表
    nameField.selectAll(); //选择文本框所有字符
}
```

- (4) 编写“抽取”按钮的事件处理方法，在该方法中把文本域保存的所有观众名称分割成字符串数组，然后通过随机数生成数组下标，当然这个下标是不固定的，然后在另一个文本



域控件中输出抽取幸运观众的颁奖信息。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {  
    String perstring = personnelArea.getText(); // 获取人员列表文本  
    String[] personnelArray = perstring.split("\n{1,}"); // 获取人员数组  
    int index = (int) (Math.random() * personnelArray.length); // 生成随机数组索引  
    // 定义包含格式参数的中奖信息  
    String formatArg = "本次抽取观众人员：\n\t%1$s\n 恭喜%1$s 成为本次观众抽奖的大奖得主。  
    + "\n\n 我们将为%1$s 颁发：\n\t过期的酸奶二十箱。";  
    // 为中奖信息添加人员参数  
    String info = String.format(formatArg, personnelArray[index]);  
    resultArea.setText(info); // 在文本域显示中奖信息  
}
```

指点迷津：

在上面的代码中应用了 String 类的 format()方法按指定的方式格式化字符串。其中该方法的第一个参数用于指定格式字符串，第二个参数用于指定格式字符串中由格式说明符引用的参数。这里为幸运观众的名称。

多学两招：

在创建与初始化数组时，通常是先定义指定类型的数组变量，然后使用 new 关键字创建数组，再分别对数组元素进行赋值。例如下面的代码：

```
int[] array = new int[3];  
array[0]=1;  
array[1]=2;  
array[2]=3;
```

另外，Java 还支持静态数组初始化，也就是在定义数组的同时为数组分配空间并赋值。例如下面的代码：

```
int[] array = { 1, 2, 3, 4};
```

技术要点

本实例的重点是把字符串中的人员名单分割为数组，以及随机生成数组的下标索引，这需要用到 String 类的 split()方法和 Math 类的 random()方法。

1. 将字符串分割为数组

String 类的 split()方法可以根据指定的正则表达式对字符串进行分割，并返回分割后的字符串数组，例如“a, b, c”，如果以“,”作为分隔符，返回值就是包含“a”、“b”和“c”3个字符串的数组。该方法的声明如下：

```
public String[] split(String regex)
```

参数说明

regex：分割字符串的定界正则表达式。

2. 生成随机数

抽奖当然是随机抽取的，这就需要用到随机数，Java 在 Math 类中提供了静态方法 random()



可以生成 0~1 之间的 double 类型的随机数值。该方法的声明如下：

```
public static double random()
```

由于该方法生成的是 0~1 之间的小数，而数组下标是整数而且又要根据数组长度来生成随机数，所以要把生成的随机数与数组长度相乘，就像本实例中的算法那样。关键代码如下：

```
int index = (int) (Math.random() * personnelArray.length); //生成随机数组索引
```

此代码把随机数与数组长度的乘积转换为整型作为随机数组下标索引。



Note

实例 034 用数组设置 JTable 表格的列名与列宽

(实例位置：配套资源\SL\05\034)

实例说明

数组在程序开发中被广泛应用，使用数组可以使程序代码更加规范，更易于维护。例如，字符串数组可用于定义表格控件的列名称，而整型数组可以用来定义列对应的宽度，本实例就通过这两个数组实现了对表格控件中表头列的设置。实例的运行效果如图 5.4 所示。

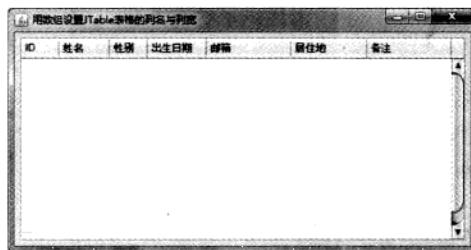


图 5.4 用数组设置 JTable 表格的列名与列宽

实现过程

(1) 在 Eclipse 中创建项目 034，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 ArrayCreateTable。在窗体中添加一个滚动面板。

(3) 编写 getTable() 方法来创建表格，在该方法中首先声明字符串数组 columns 作为表格的列名，并声明 int 类型的数组来定义每个表格列的宽度，然后创建表格的数据模型并遍历所有表格列对象，根据 int 类型数组的索引来设置表格列的宽度。关键代码如下：

```
private JTable getTable() {
    if (table == null) {
        table = new JTable(); // 创建表格
        // 定义列名数组
        String[] columns = { "ID", "姓名", "性别", "出生日期", "邮箱", "居住地", "备注" };
        // 定义列宽数组
        int[] columnWidth = { 10, 30, 10, 40, 70, 60, 70 };
        // 创建表格数据模型
        DefaultTableModel model = new DefaultTableModel(columns, 15);
        table.setModel(model); // 设置表格数据模型
        TableColumnModel columnModel = table.getColumnModel(); // 获取列模型
        int count = columnModel.getColumnCount(); // 获取列数量
    }
}
```



```
for (int i = 0; i < count; i++) { //遍历列
    TableColumn column = columnModel.getColumn(i);
    column.setPreferredWidth(columnWidth[i]); //以数组元素设置列的宽度
}
}
return table;
}
```



Note

脚下留神：

如果直接将表格控件添加到滚动面板以外的容器中，首先应该通过 JTable 类的 getTableHeader() 方法获取表格的 JTableHeader 表头类的对象，然后再将该对象添加到容器相应的位置，否则表格将没有表头，无法显示任何列名称。

技术要点

本实例的关键技术在于设置表格的数据模型和访问列模型。其中表格的数据模型可以采用 DefaultTableModel 类创建数据模型对象，而创建过程中可以把字符串数组作为参数来创建表格列的名称。

1. 创建表格数据模型

DefaultTableModel 类的构造方法有很多，其中一个可以把字符串数组作为参数来生成列名称，同时接收 int 类型的参数来设置表格添加多少行空白数据。这个构造方法的声明如下：

```
public DefaultTableModel(Object[] columnNames, int rowCount)
```

参数说明

- columnNames：存放列名的数组。
- rowCount：指定创建多少行空白数据。

2. 设置表格数据模型

JTable 类是表格控件，它提供了 setModel() 方法来设置表格的数据模型。设置数据模型以后，表格控件可以从数据模型中提取表头所有列名称和所有行数据，这个数据模型将负责表格所有数据的维护。该设置表格模型的方法的声明如下：

```
public void setModel(TableModel dataModel)
```

参数说明

dataModel：此表的新数据模型。

3. 获取表格列模型

表格中所有列对象都存放在列模型中，它们用于定义表格的每个列的名称及宽度等信息。表格的列模型可以通过 getColumnModel() 方法来获取。其方法的声明如下：

```
public TableColumnModel getColumnModel()
```

4. 设置列宽度

列对象存放在列模型中，并且列的宽度需要通过列对象的 setPreferredWidth() 方法来设置。该方法的声明如下：

```
public void setPreferredWidth(int preferredWidth)
```

参数说明

preferredWidth：列对象的首选宽度参数。



实例 035 使用按钮控件数组实现计算器界面

(实例位置: 配套资源\SL\05\035)

实例说明

控件数组的应用范围非常广泛,合理使用控件数组可以提高程序开发效率。本实例将应用按钮控件数组来管理界面中的所有按钮控件,从而使用最少的代码实现模拟的计算器界面。实例的运行效果如图 5.5 所示。

实现过程

(1) 在 Eclipse 中创建项目 035,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类,名称为 ButtonArrayExample。在窗体中添加一个文本框控件用于模拟计算器的液晶屏。

(3) 在构造方法中设置窗体标题,布局管理器,并创建 JButton 控件的二维数组,其中每个数组元素都初始化为一个按钮控件,同时再声明一个按钮名称的字符串数组,这两个数组共同初始化界面中的所有按钮控件。关键代码如下:

```
public ButtonArrayExample() {
    super(); //继承父类的构造方法
    BorderLayout borderLayout = (BorderLayout) getContentPane().getLayout();
    borderLayout.setHgap(20);
    borderLayout.setVgap(10);
    setTitle("按钮数组实现计算器界面"); //设置窗体的标题
    setBounds(100, 100, 290, 282); //设置窗体的显示位置及大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭按钮的动作作为退出
    textField = new JTextField();
    textField.setHorizontalAlignment(SwingConstants.TRAILING);
    textField.setPreferredSize(new Dimension(12, 50));
    getContentPane().add(textField, BorderLayout.NORTH);
    textField.setColumns(10);
    final GridLayout gridLayout = new GridLayout(4, 0); //创建网格布局管理器对象
    gridLayout.setHgap(5); //设置组件的水平间距
    gridLayout.setVgap(5); //设置组件的垂直间距
    JPanel panel = new JPanel(); //获得容器对象
    panel.setLayout(gridLayout); //设置容器采用网格布局管理器
    getContentPane().add(panel, BorderLayout.CENTER);
    String[][] names = { { "1", "2", "3", "+" }, { "4", "5", "6", "-" },
        { "7", "8", "9", "×" }, { ".", "0", "=", "+" } };
    JButton[][] buttons = new JButton[4][4];
    for (int row = 0; row < names.length; row++) {
        for (int col = 0; col < names.length; col++) {
            buttons[row][col] = new JButton(names[row][col]); //创建按钮对象
        }
    }
}
```



图 5.5 按钮控件数组实现计算器界面



Note



```
panel.add(buttons[row][col]);  
}  
}  
}
```



技术要点

Note 本实例的关键点在于 GridLayout 布局管理器的应用，通过它可以自动完成控件的布局与大小控制，否则，程序还要单独创建控制每个控件位置与大小的代码，其代码复杂度可想而知。通过 GridLayout 布局管理器，只需要指定布局的行列数量即可。下面介绍一下 GUI 如何使用 GridLayout 布局管理器。

1. 创建指定行列数量的布局管理器

可以在 GridLayout 类的构造方法中传递两个 int 类型的参数分别指定布局的行数与列数。其方法的声明如下：

```
public GridLayout(int rows, int cols)
```

参数说明

- rows：布局的行数。
- cols：布局的列数。

2. 设置容器的布局管理器

创建容器布局管理器后，可以把它添加到某个容器的 layout 属性中，这需要调用容器的设置布局管理器的方法来实现。其方法的声明如下：

```
public void setLayout(LayoutManager mgr)
```

参数说明

mgr：布局管理器对象。

实例 036 通过复选框控件数组实现添加多个复选框控件

(实例位置：配套资源\SL\05\036)

实例说明

复选框控件在 GUI 程序界面设计时经常使用，例如选择用户爱好的程序界面中要添加很多选项，这些选项如果通过 GUI 界面设计器来录入非常费时，而且生成的代码臃肿，不方便维护。不过可以通过复选框控件数组实现在窗体中添加多个复选框。本实例将通过复选框控件数组实现选择用户爱好信息的复选框，并且界面中的复选框数量可以根据指定复选框名称的字符串数组的长度来自动调节。实例的运行效果如图 5.6 所示。



图 5.6 通过复选框控件数组实现
添加多个复选框控件

实现过程

(1) 在 Eclipse 中创建项目 036，并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 CheckBoxArray。在窗体中添加一个标签显示“你的爱好有哪些：”。

(3) 编写 getPanel()方法创建面板并在面板中通过控件数组来创建爱好复选框。其中所有复选框的文本都是由字符串数组定义的，复选框的数量也是根据字符串数组长度确定的。关键代码如下：

```
private JPanel getPanel() {
    if (panel == null) {
        panel = new JPanel(); // 创建面板对象
        panel.setLayout(new GridLayout(0, 4)); // 设置网格布局管理器
        // 创建控件文本数组
        String[] labels = { "足球", "篮球", "魔术", "乒乓球", "看电影", "魔兽世界", "CS 战队",
            "羽毛球", "游泳", "旅游", "爬山", "唱歌", "写博客", "动物世界", "拍照", "弹吉他",
            "读报纸", "飙车", "逛街", "逛商场", "麻将", "看书", "上网看资料", "新闻", "军事",
            "八卦", "养生", "饮茶" };
        JCheckBox[] boxes = new JCheckBox[labels.length]; // 创建控件数组
        for (int i = 0; i < boxes.length; i++) { // 遍历控件数组
            boxes[i] = new JCheckBox(labels[i]); // 初始化数组中的复选框组件
            panel.add(boxes[i]); // 把数组元素（即每个复选框）添加到面板中
        }
    }
    return panel;
}
```

多学两招：

在编写一个方法时，要考虑到方法的通用性，尽量对方法进行抽象让方法适合更多的模块调用，例如本实例中的 getPanel()方法完全可以把控件标签文本数组作为方法的参数，这样其他模块就可以为该方法传递参数创建爱好选择面板了。

技术要点

本实例中应用的主要技术是在 JPanel 面板中应用复选框控件数组添加复选框控件。应用复选框控件数组添加复选框控件的具体实现步骤如下：

- (1) 定义一个字符串数组，内容为复选框的标题文本。
- (2) 创建 JCheckBox 类型的控件数组，即复选框控件数组，其长度与步骤 (1) 中创建的字符串数组的长度相同。
- (3) 通过 for 循环遍历刚刚创建的复选框控件数组，并将数组元素（即每个复选框）添加到面板中。

实例 037 使用选择排序法对数组排序

(实例位置：配套资源\SL\05\037)

实例说明

选择排序 (Selection Sort) 是一种简单直观的排序算法。本实例演示如何使用选择排序法





对一维数组进行排序，运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本域控件中；然后单击“排序”按钮，使用选择排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本域控件中。实例的运行效果如图 5.7 所示。

实现过程

- (1) 在 Eclipse 中创建项目 037，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 SelectSort。在窗体中添加两个文本域控件和“生成随机数组”、“排序”两个按钮。
- (3) 编写“生成随机数组”按钮的事件处理方法，在该方法中创建 Random 随机数对象，初始化数组元素值时，通过该对象为每个数组元素生成随机数。关键代码如下：

```
private int[] array = new int[10];
protected void do_button_actionPerformed(ActionEvent e) {
    Random random = new Random(); // 创建随机数对象
    textArea1.setText(""); // 清空文本域
    for (int i = 0; i < array.length; i++) { // 初始化数组元素
        array[i] = random.nextInt(50); // 生成 50 以内的随机数
        textArea1.append(array[i] + " "); // 把数组元素显示在文本域控件中
    }
}
```

- (4) 编写“排序”按钮的事件处理方法，在该方法中使用排序算法对生成的随机数组进行排序，然后把排序后的数组元素显示到文本域控件中。关键代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    textArea2.setText(""); // 清空文本域
    int index;
    for (int i = 1; i < array.length; i++) {
        index = 0;
        for (int j = 1; j <= array.length - i; j++) {
            if (array[j] > array[index]) {
                index = j; // 查找最大值
            }
        }
        // 交换在 array.length-i 和 index (最大值) 位置的两个数
        int temp = array[array.length - i];
        array[array.length - i] = array[index];
        array[index] = temp;
    }
    for (int i = 0; i < array.length; i++) {
        textArea2.append(array[i] + " "); // 把排序后的数组元素显示到文本域中
    }
}
```



图 5.7 使用选择排序法对一维数组排序

**多学两招：**

利用选择排序法从数组中挑选最大值并放在数组最后，而遇到重复的相等值不会做任何处理，所以如果程序允许数组有重复值的情况，建议使用选择排序方法，因为它的数据交换次数较少，相对速度也会略微提升，这取决于数组中重复值的数量。

**Note****技术要点**

本实例应用的主要技术点就是选择排序算法。选择排序算法的基本思想如下：

每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。例如，对一个一维数组采用选择排序算法进行排序的过程如图 5.8 所示。

```

举例：

初始数组资源【63 4 24 1 3 15】

第一趟排序后【15 4 24 1 3】63
第二趟排序后【15 4 3 1】24 63
第三趟排序后【1 4 3】15 24 63
第四趟排序后【1 3】4 15 24 63
第五趟排序后【1】3 4 15 24 63

```

图 5.8 对一个一维数组采用选择排序算法进行排序的过程

实例 038 使用冒泡排序法对数组排序

（实例位置：配套资源\SL\05\038）

实例说明

本实例演示如何使用冒泡排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本域控件中；然后单击“排序”按钮，使用冒泡排序法对生成的一维数组进行排序，并将排序过程中一维数组的变化显示在下方的文本域控件中。实例的运行效果如图 5.9 所示。



图 5.9 使用冒泡排序法对一维数组排序



实现过程



Note

(1) 在 Eclipse 中创建项目 038，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 BubbleSort。在窗体中添加两个文本域控件和“生成随机数组”、“排序”两个按钮。

(3) 编写“生成随机数组”按钮的事件处理方法，在该方法中创建 Random 随机数对象，初始化数组元素值时，通过该对象为每个数组元素生成随机数。关键代码如下：

```
private int[] array = new int[10];
protected void do_buttonActionPerformed(ActionEvent e) {
    Random random = new Random();           // 创建随机数对象
    textArea1.setText("");                  // 清空文本域
    for (int i = 0; i < array.length; i++) {   // 初始化数组元素
        array[i] = random.nextInt(50);       // 生成 50 以内的随机数
        textArea1.append(array[i] + " ");      // 把数组元素显示在文本域控件中
    }
}
```

(4) 编写“排序”按钮的事件处理方法，在该方法中使用排序算法对生成的随机数组进行排序，然后把排序后的数组元素显示到文本域控件中。关键代码如下：

```
protected void do_button_1ActionPerformed(ActionEvent e) {
    textArea2.setText("");                  // 清空文本域
    for (int i = 1; i < array.length; i++) {   // 比较相邻两个元素，较大的数往后冒泡
        for (int j = 0; j < array.length - i; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];           // 把第一个元素值保存到临时变量中
                array[j] = array[j + 1];         // 把第二个元素值保存到第一个元素单元中
                array[j + 1] = temp;           // 把临时变量也就是第一个元素原值保存到第二个元素中
            }
            textArea2.append(array[j] + " ");  // 把排序后的数组元素显示到文本域中
        }
        textArea2.append("【");
        for (int j = array.length - i; j < array.length; j++) {
            textArea2.append(array[j] + " ");  // 把排序后的数组元素显示到文本域中
        }
        textArea2.append("】\n");
    }
}
```

多学两招：

实际上，初始化数组时可以省略 new 运算符和数组的长度，编译器将根据初始值的数量来自动计算数组长度，并创建数组。例如 int[] array = {1, 2, 3, 4, 5};。

技术要点

在实现本实例时，主要用到了冒泡排序算法。冒泡排序的基本思想是对比相邻的元素值，如果满足条件就交换元素值，把较小的元素移动到数组前面，把大的元素移动到数组后面（也就是交换两个元素的位置），这样数组元素就像气泡一样从底部上升到顶部。



冒泡算法在双层循环中实现，其中外层循环控制排序轮数，是要排序数组长度-1 次。而内层循环主要是用于对比临近元素的大小，以确定是否交换位置，对比和交换次数依排序轮数而减少。例如一个拥有 6 个元素的数组，在排序过程中每一次循环的排序过程和结果如图 5.10 所示。

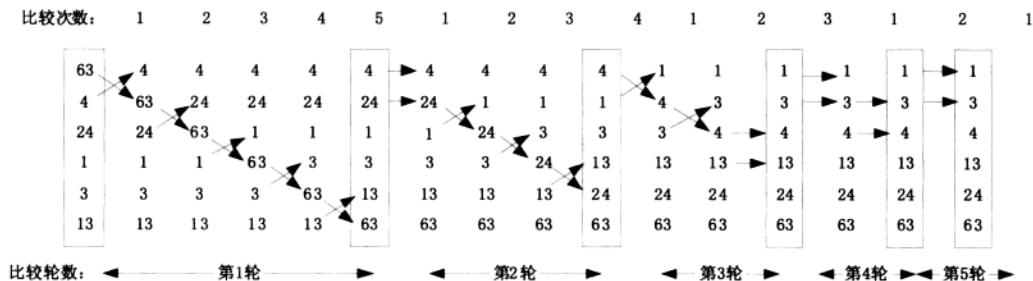
**Note**

图 5.10 排序过程和结果

第一轮外层循环时把最大的元素值 63 移动到了最后面（相应地比 63 小的元素向前移动，类似气泡上升），第二轮外层循环不再对比最后一个元素值 63，因为它已经确认为最大（不需要上升），应该放在最后，需要对比和移动的是其他剩余元素，这次将元素 24 移动到了 63 的前一个位置。其他循环将依此类推，继续完成排序任务。

实例 039 使用快速排序法对数组排序

(实例位置: 配套资源\SL\05\039)

实例说明

快速排序 (Quick Sort) 是对气泡排序的一种改进，其排序速度相对较快。本实例演示如何使用快速排序法对一维数组进行排序，运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“排序”按钮，使用快速排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例的运行效果如图 5.11 所示。



图 5.11 使用快速排序法对数组排序



实现过程

- (1) 在 Eclipse 中创建项目 039，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 Quick Sort。在窗体中添加一个文本框、一个文本域控件和“生成随机数组”、“排序”两个按钮。

Note

指点迷津：

由于“生成随机数组”按钮的事件处理方法与实例 038 中的“生成随机数组”按钮的事件处理方法相同，都是用于生成一个由多个随机数组成的一维数组，所以这里不给出具体的实现方法。

- (3) 编写“排序”按钮的事件处理方法，在该方法中利用快速排序算法对生成的随机数组进行排序，并将排序过程输出到文本域控件中。关键代码如下：

```
protected void do_button_1ActionPerformed(ActionEvent e) {
    textArea2.setText("");
    quickSort(array, 0, array.length - 1);
}
```

- (4) 编写快速排序方法 quickSort()，这个方法将被按钮的事件处理方法调用，该方法在实现快速排序的同时，把排序过程显示到文本域控件中。关键代码如下：

```
private void quickSort(int sortarray[], int lowIndex, int highIndex) {
    int lo = lowIndex; //记录最小索引
    int hi = highIndex; //记录最大索引
    int mid; //记录分界点元素
    if (highIndex > lowIndex) {
        mid = sortarray[(lowIndex + highIndex) / 2]; //确定中间分界点元素值
        while (lo <= hi) {
            while ((lo < highIndex) && (sortarray[lo] < mid)) //确定不大于分界元素值的最小索引
                ++lo;
            while ((hi > lowIndex) && (sortarray[hi] > mid)) //确定大于分界元素值的最大索引
                --hi;
            if (lo <= hi) { //如果最小索引与最大索引没有重叠
                swap(sortarray, lo, hi); //交换两个索引的元素
                ++lo; //递增最小索引
                --hi; //递减最大索引
            }
        }
        if (lowIndex < hi) //递归排序没有未分解元素
            quickSort(sortarray, lowIndex, hi);
        if (lo < highIndex) //递归排序没有未分解元素
            quickSort(sortarray, lo, highIndex);
    }
}
```

- (5) 由于快速排序方法中频繁地交换数组元素，而且在程序代码中出现的次数较多，所以应该把数组元素交换单独提炼为一个 swap() 方法，以实现代码重用，并且可以在该方法中掌



握排序过程并显示到文本域控件。关键代码如下：

```
private void swap(int swapArray[], int i, int j) {
    int temp = swapArray[i];
    swapArray[i] = swapArray[j];
    swapArray[j] = temp;
    for (int k = 0; k < array.length; k++) {           //把数组元素显示到文本域
        textArea2.append(array[k] + " ");
    }
    textArea2.append("\n");                            //追加换行符
}
```

技术要点

本实例主要用到了快速排序算法，下面对快速排序算法的实现原理进行详细讲解。

快速排序算法是对冒泡排序算法的一种改进，它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据小，然后再按此方法对这两部分数据分别进行快速排序。整个排序过程可以递归进行，以此使整个数据变成有序序列。

假设要排序的数组是 A[1]…A[N]，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，递归调用此过程，即可实现数组的快速排序。

使用快速排序法的排序过程如图 5.12 所示。

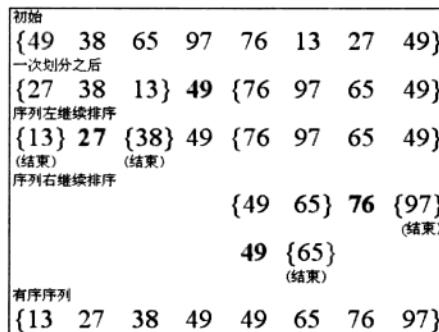


图 5.12 快速排序法的排序过程

实例 040 使用直接插入法对数组排序

(实例位置：配套资源\SL\05\040)

实例说明

本实例演示如何使用直接插入排序法对一维数组进行排序。运行本实例，首先单击“随机生成数组”按钮，生成一个随机数组，并显示在左边的文本框中；然后单击“排序”按钮，使用直接插入排序法对生成的一维数组进行排序，并将排序后的一维数组显示在右边的文本框中。实例的运行效果如图 5.13 所示。





Note

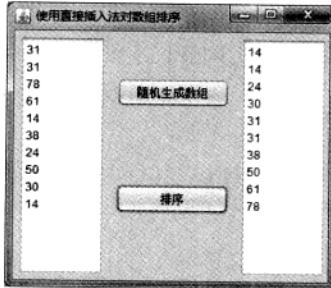


图 5.13 使用直接插入法对一维数组进行排序

实现过程

- (1) 在 Eclipse 中创建项目 040，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 InsertSort。在窗体中添加两个文本域控件和“随机生成数组”、“排序”两个按钮。
- (3) 编写“随机生成数组”按钮的事件处理方法，在该方法中利用 Random 类的实例对象的 nextInt() 方法生成随机数，并为数组设置初始值。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {  
    Random random = new Random(); // 创建随机数对象  
    textArea1.setText("");  
    for (int i = 0; i < array.length; i++) { // 初始化数组元素  
        array[i] = random.nextInt(90); // 生成 50 以内的随机数  
        textArea1.append(array[i] + "\n"); // 把数组元素显示在文本域控件中  
    }  
}
```

- (4) 编写“排序”按钮的事件处理方法，在该方法中使用直接插入排序算法对数组进行排序，并将排序后的数组显示到界面中。关键代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {  
    int tmp; // 定义临时变量  
    int j;  
    for (int i = 1; i < array.length; i++) {  
        tmp = array[i]; // 保存临时变量  
        for (j = i - 1; j >= 0 && array[j] > tmp; j--) {  
            array[j + 1] = array[j]; // 数组元素交换  
        }  
        array[j + 1] = tmp; // 在排序位置插入数据  
    }  
    textArea2.setText("");  
    for (int i = 0; i < array.length; i++) { // 初始化数组元素  
        textArea2.append(array[i] + "\n"); // 把数组元素显示在文本域控件中  
    }  
}
```

技术要点

本实例主要用到了直接插入排序算法。下面对直接插入排序算法的实现原理进行详细讲解。插入排序是将一个记录插入到有序数列中，使得到的新数列仍然有序。插入排序算法的思想



想是：将 n 个有序数存放在数组 a 中，要插入的数为 x ，首先确定 x 插在数组中的位置 p ，数组中 p 之后的元素都向后移一个位置，空出 $a(p)$ ，将 x 放入 $a(p)$ 。这样即可实现插入后数列仍然有序。

使用插入排序法排序的过程如图 5.14 所示。



Note

	0	1	2	3	4	5	6	
k=2	20	40	90	30	80	70	50	
	0	1	2	3	4	5	6	
k=3	20	30	40	90	80	70	50	
	0	1	2	3	4	5	6	
k=4	20	30	40	80	90	70	50	
	0	1	2	3	4	5	6	
k=5	20	30	40	70	80	90	50	
	0	1	2	3	4	5	6	
k=6	20	30	40	50	70	80	90	
								n=7

图 5.14 直接插入排序法排序过程

实例 041 使用 sort()方法对数组排序

(实例位置：配套资源\SL\05\041)

实例说明

实际开发项目时，经常需要在程序中对数组进行排序，而且，在计算机常用算法中，也提供了很多种对数组进行排序的算法，如冒泡排序法、直接插入法和选择排序法等，但在使用排序算法时，开发人员必须手动编写一堆代码，而且有的实现起来比较麻烦。Java 的 Arrays 类提供了一个 sort()方法，使用该方法可以很方便地对各种数组进行排序，大大降低了数组排序的难度，本实例就将使用该方法对数组进行快速排序。实例的运行效果如图 5.15 所示。

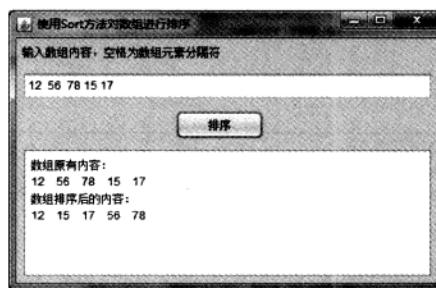


图 5.15 使用 sort()方法对数组进行快速排序

实现过程

- (1) 在 Eclipse 中创建项目 041，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 InsertSort。在窗体中添加一个文本框、一个文本域控件和一个“排序”按钮。



(3) 为“排序”按钮编写事件处理方法，在该方法中要接收用户的输入字符串，并以字符串中的空格为分割符，将字符串转换为字符串数组，再把字符串数组转换为整数数组，然后调用 Arrays 类的 sort()方法对其进行排序，并显示到窗体中。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {  
    String text = arrayField.getText(); // 获取用户输入  
    text=text.trim(); // 去除首尾空格  
    if("".equals(text)){  
        JOptionPane.showMessageDialog(null,"请输入要排序的数组内容");  
        return;  
    }  
    String[] arrayStr = text.split(" {1,}"); // 拆分输入为数组  
    int[] array = new int[arrayStr.length]; // 创建整数类型数组  
    sortArea.setText("数组原有内容: \n");  
    for (String string : arrayStr) { // 输出原有数组内容  
        sortArea.append(string + "      ");  
    }  
    for (int i = 0; i < array.length; i++) { // 初始化整型数组  
        array[i] = Integer.parseInt(arrayStr[i]);  
    }  
    sortArea.append("\n");  
    Arrays.sort(array); // 使用 sort() 方法对整型数组进行排序  
    sortArea.append("数组排序后的内容: \n");  
    for (int value : array) { // 输出排序后的数组内容  
        sortArea.append(value + "      ");  
    }  
}
```

(4) 编写文本框的按键事件处理方法，通过该方法的编写来限制文本框可输入的字符，当用户按下非数字与空格字符时，可取消本次输入的有效性。关键代码如下：

```
protected void do_arrayField_keyPressed(KeyEvent e) {  
    char key = e.getKeyChar(); // 获取用户按键字符  
    String mask = "0123456789 " + (char) 8; // 定义规范化字符模板  
    if (mask.indexOf(key) == -1) { // 判断按键字符是否属于规范化字符范围  
        e.consume(); // 取消非规范化字符的输入有效性  
    }  
}
```

指点迷津：

Arrays 类提供了创建、操作、搜索和排序数组的方法。在程序开发中有效利用 Arrays 类的各种方法来完成数组操作将大幅度提升程序开发效率，并且 Arrays 类的方法是经过测试的，可以减少程序开发中错误代码的出现。

技术要点

本实例在对数组进行快速排序时，主要用到了 Arrays 类的 sort() 方法，下面对其进行详细讲解。

Arrays 类位于 java.util 包，它是数组的一个工具类，包含很多方法，其中 sort() 方法就是 Arrays 类提供的对数组进行排序的方法，它有很多重载格式，可以接收任何数据类型的数组并



执行不同类型的排序。本实例使用 sort()方法的 int 参数类型的重载实现，其方法声明如下：

```
public static void sort(int[] array)
```

参数说明

array：要排序的 int 类型的一维数组。



Note

实例 042 反转数组中元素的顺序

(实例位置：配套资源\SL\05\042)

实例说明

反转数组就是以相反的顺序把原有数组的内容重新排序。本实例在 GUI 窗体中演示反转数组中元素的顺序。运行本实例，首先在界面的文本框内输入数组元素，每个元素使用空格分隔，然后单击界面上的“反转数组元素”按钮，程序将对数组进行反转运算，并把运算过程中对数组的改变显示在窗体中。实例的运行效果如图 5.16 所示。

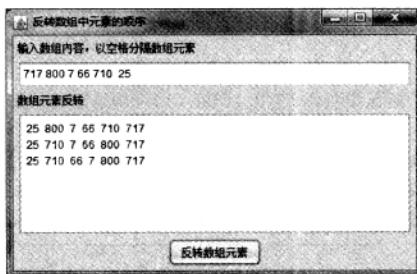


图 5.16 反转数组中元素的顺序

实现过程

- (1) 在 Eclipse 中创建项目 042，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 ReverseSort。在窗体中添加一个文本框、一个文本域控件和“反转数组元素”按钮。
- (3) 编写“反转数组元素”按钮的事件处理方法，在该方法中获取用户输入的字符串，然后以空格为分隔符，把字符串拆分成字符串数组，最后在通过反转算法排序数组的同时，把反转过程中数组的变化输出到文本域控件中。关键代码如下：

```
protected void do_button_1ActionPerformed(ActionEvent e) {
    String inText = textField.getText(); // 获取用户输入
    String[] array = inText.split(" {1,}"); // 把字符串分割为数组
    int len = array.length; // 获取数组长度
    textArea.setText(""); // 清空文本域控件内容
    for (int i = 0; i < len / 2; i++) { // 反转数组元素
        String temp = array[i];
        array[i] = array[len - 1 - i];
        array[len - 1 - i] = temp;
        for (String string : array) { // 在文本域中显示数组排序过程
            textArea.append(string + " ");
        }
    }
}
```



```
        }
        textArea.append("\n");           //追加换行符
    }
}
```

脚下留神：

“反转”并不等于“倒序排序”。“反转”只是将数组元素的顺序进行颠倒，并不对其执行排序操作；而“倒序排序”则是对数组中的元素进行从大到小的排序。

技术要点

本实例的核心技术是使用了数组反转算法。反转算法的基本思想比较简单，也很好理解，思路就是：把数组最后一个元素与第一个元素替换，倒数第二个元素与第二个元素替换，依此类推，直到把所有数组元素反转替换。

反转数组元素是对数组两边的元素进行替换，所以只需要循环数组长度的半数。例如，对长度为 6 的一维数组进行反转，只需要循环 6/2，即 3 次，具体的反转过程如图 5.17 所示。

举例：

初始数组资源 【10 20 30 40 50 60】

第一趟排序后	60	【20	30	40	50】	10
第二趟排序后	60	50	【30	40】	20	10
第三趟排序后	60	50	40	30	20	10

图 5.17 对长度为 6 的数组进行反转的过程

第 6 章

面向对象入门

本章读者可以学到如下实例：

- ▶ 实例 043 自定义图书类
- ▶ 实例 044 温度单位转换工具
- ▶ 实例 045 成员变量的默认初始化值
- ▶ 实例 046 单例模式的应用
- ▶ 实例 047 汉诺塔问题求解
- ▶ 实例 048 编写同名的方法
- ▶ 实例 049 构造方法的应用
- ▶ 实例 050 统计图书的销售量
- ▶ 实例 051 两只完全相同的宠物
- ▶ 实例 052 重新计算对象的哈希码
- ▶ 实例 053 使用字符串输出对象
- ▶ 实例 054 Java 对象的假克隆
- ▶ 实例 055 Java 对象的浅克隆
- ▶ 实例 056 Java 对象的深克隆
- ▶ 实例 057 序列化与对象克隆
- ▶ 实例 058 深克隆效率的比较



实例 043 自定义图书类

(实例位置: 配套资源\SL\06\043)



Note

实例说明

在使用 Java 语言进行开发时,时刻要以面向对象的思想考虑问题。面向对象的基础就是类,本实例将演示如何自定义类,它用于表示图书。实例的运行效果如图 6.1 所示。

实现过程

(1) 在 Eclipse 中创建项目 043,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Book。在 Book 类中定义 3 个成员变量,分别表示书名、作者和价格,同时提供构造方法和成员方法来修改成员变量。关键代码如下:

```
public class Book {  
    private String title; // 定义书名  
    private String author; // 定义作者  
    private double price; // 定义价格  
    public Book(String title, String author, double price) { // 利用构造方法初始化域  
        this.title = title;  
        this.author = author;  
        this.price = price;  
    }  
    public String getTitle() { // 获得书名  
        return title;  
    }  
    public String getAuthor() { // 获得作者  
        return author;  
    }  
    public double getPrice() { // 获得价格  
        return price;  
    }  
}
```

(3) 在 com.mingrisoft 包中再创建类文件,名称为 Test。在该类的 main()方法中,创建一个 Book 对象并输出其属性。关键代码如下:

```
public class Test {  
    public static void main(String[] args) {  
        Book book = new Book("《Java 从入门到精通 (第 2 版)》", "明日科技", 59.8); // 创建对象  
        System.out.println("书名: " + book.getTitle()); // 输出书名  
        System.out.println("作者: " + book.getAuthor()); // 输出作者  
        System.out.println("价格: " + book.getPrice() + "元"); // 输出价格  
    }  
}
```

(4) 编译并运行 Test 文件,在 Eclipse 控制台上的输出效果如图 6.1 所示。

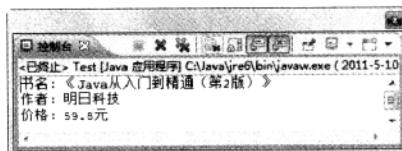


图 6.1 输出自定义的图书类对象



技术要点

在 Java 中，使用 `class` 关键字来定义类。一个类，通常包括域和方法两部分。域表示对象的状态，方法表示对象的行为。使用 `new` 关键字可以创建类的对象。通常情况下，不同的对象状态是有差别的。可以使用构造方法在创建对象时就设置状态，也可以使用方法在创建对象后修改对象的状态。



Note

实例 044 温度单位转换工具

(实例位置：配套资源\SL\06\044)

实例说明

目前有两种广泛使用的温度单位，即摄氏度和华氏度。在标准大气压下，沸腾的水可以表示成 100 摄氏度和 212 华氏度。本实例将编写一个简单的温度单位转换工具，它可以将用户输入的摄氏度转换成华氏度，其运行效果如图 6.2 所示。

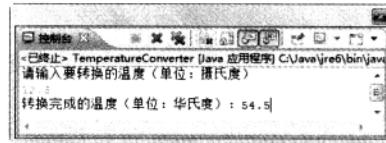


图 6.2 将输入的摄氏度温度转换成华氏度

实现过程

(1) 在 Eclipse 中创建项目 044，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `CelsiusConverter`。在 `CelsiusConverter` 类中定义了两个方法，`getFahrenheit()`方法用于将摄氏温度转换成华氏温度，`main()`方法用于测试。关键代码如下：

```
public class CelsiusConverter {
    public double getFahrenheit(double celsius) {
        double fahrenheit = 1.8 * celsius + 32;           //计算华氏温度
        return fahrenheit;                                //返回华氏温度
    }
    public static void main(String[] args) {
        System.out.println("请输入要转换的温度(单位: 摄氏度)");
        Scanner in = new Scanner(System.in);               //获得控制台输入
        double celsius = in.nextDouble();                  //获得用户输入的摄氏温度
        CelsiusConverter converter = new CelsiusConverter(); //创建类的对象
        double fahrenheit = converter.getFahrenheit(celsius); //转换温度为华氏度
        System.out.println("转换完成的温度(单位: 华氏度): " + fahrenheit); //输出转换结果
    }
}
```

技术要点

本实例中定义了一个普通方法 `getFahrenheit()`，它完成了将摄氏度转换成华氏度的功能。在 Java 中使用普通方法需要先创建一个方法所在类的对象，然后通过这个对象调用这个方法。这点与静态方法有明显不同。后者可以直接使用类名调用。



实例 045 成员变量的默认初始化值

(实例位置: 配套资源\SL\06\045)



Note

实例说明

一般来说, 变量在使用之前需要对其进行初始化。Java 中的成员变量有些特殊, 虚拟机会自动为其进行初始化。为了了解默认的初始化是否符合编程的需求, 本实例将输入 Java 中各种类型的成员变量的初始化值, 其运行效果如图 6.3 所示。

实现过程

(1) 在 Eclipse 中创建项目 045, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Initialization。在 Initialization 方法中定义了 8 种基本类型的成员变量和一个引用类型的成员变量, 在未初始化的情况下, 直接在 main() 方法中将其输出。关键代码如下:

```
public class Initialization {
    private byte b; // 声明比特类型变量 b
    private short s; // 声明短整型类型变量 s
    private int i; // 声明整型类型变量 i
    private long l; // 声成长整型类型变量 l
    private float f; // 声明清精度浮点类型变量 f
    private double d; // 声明双精度浮点类型变量 d
    private boolean bl; // 声明布尔类型变量 bl
    private char c; // 声明字符类型变量 c
    private String string; // 声明引用类型变量 string

    public static void main(String[] args) {
        Initialization init = new Initialization();
        System.out.println("比特类型的初始值: " + init.b); // 输出比特类型变量初始值
        System.out.println("短整型类型的初始值: " + init.s); // 输出短整型类型变量初始值
        System.out.println("整型类型的初始值: " + init.i); // 输出整型类型变量初始值
        System.out.println("长整型类型的初始值: " + init.l); // 输出长整型类型变量初始值
        System.out.println("单精度浮点类型的初始值: " + init.f); // 输出单精度浮点类型变量初始值
        System.out.println("双精度浮点类型的初始值: " + init.d); // 输出双精度浮点类型变量初始值
        System.out.println("布尔类型的初始值: " + init.bl); // 输出布尔类型变量初始值
        System.out.println("字符类型的初始值: " + init.c); // 输出字符类型变量初始值
        System.out.println("引用类型的初始值: " + init.string); // 输出引用类型变量初始值
    }
}
```

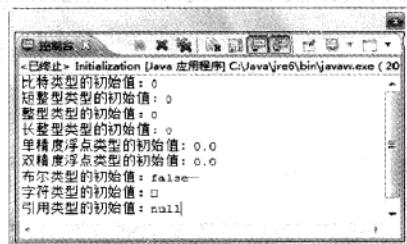


图 6.3 成员变量的默认初始化值

脚下留神:

对于引用类型的变量, 在使用之前需要进行初始化, 否则会抛出 NullPointerException 异常。



技术要点

除了成员变量，Java 中还可以定义局部变量。局部变量在使用之前必须要对其进行初始化，虚拟机不会进行初始化工作；否则会报告错误。



实例 046 单例模式的应用

(实例位置：配套资源\SL\06\046)

Note

实例说明

中国历史上的皇帝通常仅有一人。为了保障其唯一性，古人采用增加“防伪标识”的办法，如玉玺。更简单的方法是限制皇帝的创建。本实例使用单例模式来保证皇帝的唯一性。实例的运行效果如图 6.4 所示。

实现过程

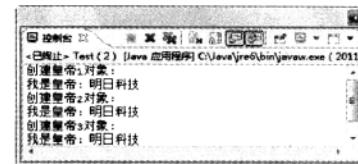


图 6.4 使用单例模式保证皇帝的唯一性

(1) 在 Eclipse 中创建项目 046，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Emperor。在 Emperor 类中，将构造方法设置成私有，并提供一个静态方法用于获得该类的实例。关键代码如下：

```
public class Emperor {
    private static Emperor emperor = null;
    private Emperor() {                                //声明一个 Emperor 类的引用
        }                                              //将构造方法私有
    public static Emperor getInstance() {             //实例化引用
        if (emperor == null) {
            emperor = new Emperor();
        }
        return emperor;
    }
    public void getName() {                          //使用普通方法输出皇帝的名字
        System.out.println("我是皇帝：明日科技");
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类的 main()方法中，创建 3 个 Emperor 对象并输出了其名字。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("创建皇帝 1 对象：");
        Emperor emperor1 = Emperor.getInstance();      //创建皇帝对象
        emperor1.getName();                           //输出皇帝的名字
        System.out.println("创建皇帝 2 对象：");
        Emperor emperor2 = Emperor.getInstance();      //创建皇帝对象
        emperor2.getName();                           //输出皇帝的名字
        System.out.println("创建皇帝 3 对象：");
    }
}
```



```

Emperor emperor3 = Emperor.getInstance();
emperor3.getName(); //创建皇帝对象
}
}

```



技术要点

单例模式的特点在于仅能获得一个对象。为了防止其他用户创建对象，需要将构造方法设置成 private 的，然后提供一个静态方法，该方法返回这个类的对象。

实例 047 汉诺塔问题求解

(实例位置：配套资源\SL\06\047)

实例说明

汉诺塔问题的描述如下：有 A、B 和 C 3 根柱子，在 A 上从下往上按照从小到大的顺序放着 64 个圆盘，以 B 为中介，把盘子全部移动到 C 上。移动过程中，要求任意盘子的下面要么没有盘子，要么只能有比它大的盘子。本实例将演示如何求解 3 阶汉诺塔问题，其运行效果如图 6.5 所示。

实现过程

- (1) 在 Eclipse 中创建项目 047，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建类文件，名称为 HanoiTower。在 HanoiTower 类中定义了一个 moveDisk()方法，它使用递归算法完成汉诺塔问题的求解；一个 main()方法用于测试。关键代码如下：

```

public class HanoiTower {
    public static void moveDisk(int level, char from, char inter, char to) {
        if (level == 1) { //如果只有一个盘子就退出迭代
            System.out.println("从 " + from + " 移动盘子 1 号到 " + to);
        } else { //如果有大于一个盘子就继续迭代
            moveDisk(level - 1, from, to, inter);
            System.out.println("从 " + from + " 移动盘子 " + level + " 号到 " + to);
            moveDisk(level - 1, inter, from, to);
        }
    }
    public static void main(String[] args) {
        int nDisks = 3; //设置汉诺塔为 3 阶
        moveDisk(nDisks, 'A', 'B', 'C'); //实现移动算法
    }
}

```

技术要点

为了将 N 个盘子从 A 移动到 C，需要先将第 N 个盘子上面的 N-1 个盘子移动到 B 上，这

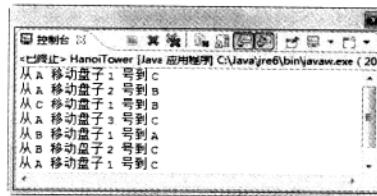


图 6.5 3 阶汉诺塔解决步骤



样才能将第 N 个盘子移动到 C 上。同理，为了将第 N-1 个盘子从 B 移动到 C 上，需要将 N-2 个盘子移动到 A 上，这样才能将第 N-1 个盘子移动到 C 上。通过递归就可以实现汉诺塔问题的求解。

实例 048 编写同名的方法

(实例位置：配套资源\SL\06\048)



Note

实例说明

在 C 语言中，如果要在同一个文件内定义两个同名的方法会报告错误。这对于编写具有相似功能不同参数的方法非常不利。在 Java 中，可以利用重载技术来完成这个需求。本实例将在一个类中定义两个同名的方法，它们的参数不同，其运行效果如图 6.6 所示。

实现过程

- (1) 在 Eclipse 中创建项目 048，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建类文件，名称为 OverloadingTest。在 OverloadingTest 类中，定义了两个同名方法 info()，但是参数不同；一个 main() 方法用于测试。关键代码如下：

```
public class OverloadingTest {
    public void info() { // 定义没有参数的 info()方法
        System.out.println("普通方法：明日科技 1 岁了！");
    }
    public void info(int age) { // 定义包含整型参数的 info()方法
        System.out.println("重载方法：明日科技" + age + "岁了！");
    }
    public static void main(String[] args) {
        OverloadingTest ot = new OverloadingTest(); // 创建 OverloadingTest 类对象
        ot.info(); // 测试无参数 info()方法
        for (int i = 1; i < 5; i++) { // 测试有参数 info()方法
            ot.info(i);
        }
    }
}
```

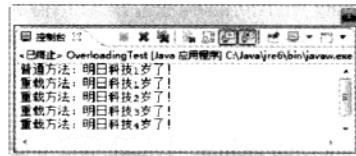


图 6.6 调用普通方法和重载方法的输出

技术要点

在 Java 中，可以通过重载（overloading）来减少方法名称的个数。当对象在调用方法时，可以根据方法参数的不同来确定执行哪个方法。方法参数的不同包括参数类型不同、参数个数不同和参数顺序不同。需要注意的是，不能通过方法的返回值来区分方法，即不能有两个方法签名相同但返回值不同的方法。

指点迷津：

方法名称和方法参数统称为方法签名。



实例 049 构造方法的应用

(实例位置: 配套资源\SL\06\049)



Note 实例说明

Java 程序的各种功能是通过对对象调用相关方法完成的,因此必须先获得对象。使用构造方法来获得对象是一种常见方式。构造方法也支持重载,本实例将演示使用不同的构造方法来获得对象,其运行效果如图 6.7 所示。

实现过程

(1) 在 Eclipse 中创建项目 049,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Person。在 Person 类中,定义了 3 个成员变量分别表示人的姓名、性别和年龄。此外还提供了构造方法和普通方法来修改成员变量的值。关键代码如下:

```
public class Person {  
    private String name; // 定义姓名  
    private String gender; // 定义性别  
    private int age; // 定义年龄  
    public Person() { // 定义没有参数的构造方法  
        System.out.println("使用无参构造方法创建对象");  
    }  
    public Person(String name, String gender, int age) { // 利用构造方法初始化域  
        this.name = name;  
        this.gender = gender;  
        this.age = age;  
        System.out.println("使用有参构造方法创建对象");  
    }  
    public String getName() { // 获得姓名  
        return name;  
    }  
    public String getGender() { // 获得性别  
        return gender;  
    }  
    public int getAge() { // 获得年龄  
        return age;  
    }  
}
```

(3) 在 com.mingrisoft 包中再创建类文件,名称为 Test。在该类的 main()方法中,创建了两个 Person 对象并输出了其属性。代码如下:

```
public class Test {  
    public static void main(String[] args) {
```

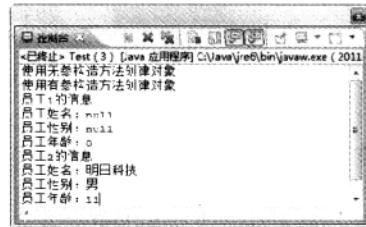


图 6.7 输出创建的两个对象



```

Person person1 = new Person();
Person person2 = new Person("明日科技", "男", 11);
System.out.println("员工 1 的信息");
System.out.println("员工姓名: " + person1.getName());           // 创建对象
System.out.println("员工性别: " + person1.getGender());        // 创建对象
System.out.println("员工年龄: " + person1.getAge());          // 输出姓名
System.out.println("员工 2 的信息");
System.out.println("员工姓名: " + person2.getName());           // 输出性别
System.out.println("员工性别: " + person2.getGender());        // 输出年龄
System.out.println("员工年龄: " + person2.getAge());          // 输出年龄
}
}

```

技术要点

构造方法是一种特殊类型的方法，它可以用来实现成员变量的初始化操作。在声明时必须遵守如下规定：

- 构造方法的名称与类名相同。
 - 构造方法没有返回值，并不是返回 void。
 - 构造方法总是与 new 操作符一起使用，即不能用对象调用构造方法。
- 此外，在构造方法中，还可以使用 this 来调用其他构造方法，使用 super 调用超类构造方法。

实例 050 统计图书的销售量

(实例位置：配套资源\SL\06\050)

实例说明

在商品（类的实例）的销售过程中，需要对销量进行统计。本实例将在类的构造方法中增加计数器来实现该功能，其运行效果如图 6.8 所示。

实现过程

(1) 在 Eclipse 中创建项目 050，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Book。在 Book 类中，定义了一个静态的成员变量用于保存实例化的次数。在构造方法中实现了计数器的功能，关键代码如下：

```

public class Book {
    private static int counter = 0;                         // 定义一个计数器
    public Book(String title) {                             // 输出书名
        System.out.println("售出图书: " + title);           // 计数器加 1
        counter++;                                         // 获得计数器的结果
    }
    public static int getCounter() {                         // 获得计数器的结果
        return counter;
    }
}

```

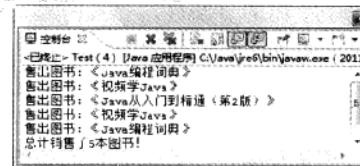


图 6.8 输出图书销售信息



Note



(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在 Test 类中定义 main()方法用于创建 Book 类对象并输出创建对象的个数。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        //创建书名数组
        String[] titles = { "《Java 从入门到精通（第 2 版）》", "《Java 编程词典》", "《视频学 Java》" };
        for (int i = 0; i < 5; i++) {
            new Book(titles[new Random().nextInt(3)]);
        }
        System.out.println("总计销售了" + Book.getCounter() + "本图书！");
    }
}
```



Note

指点迷津：

new Random().nextInt(3) 代码用于获得 0~2 的整数。

技术要点

对于非 static 成员变量，不同的对象可以对其任意修改而不会对其他对象产生影响。对于 static 成员变量，则是所有对象所共享的。任何一个对象的修改会影响其他对象。

实例 051 两只完全相同的宠物

(实例位置：配套资源\SL\06\051)

实例说明

由于生命的复杂性，寻找两只完全相同的宠物（类的对象）是不可能的。在 Java 中，可以通过比较对象的成员变量来判断对象是否相同。本实例将创建 3 只宠物猫，通过比较它们的名字、年龄、重量和颜色属性来看它们是否相同。实例的运行效果如图 6.9 所示。

实现过程

(1) 在 Eclipse 中创建项目 051，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色，并提供构造方法来设置这些属性值。本类的重点内容在于重写 equals() 和 toString() 方法。重写 equals() 方法可以比较两个对象是否相同，重写 toString() 方法可以直接输出对象。关键代码如下：

```
public class Cat {
    private String name; //表示猫咪的名字
    private int age; //表示猫咪的年龄
    private double weight; //表示猫咪的重量
```



图 6.9 输出对象信息和比较结果



Note

```

private Color color; //表示猫咪的颜色
public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性
    this.name = name;
    this.age = age;
    this.weight = weight;
    this.color = color;
}
@Override
public boolean equals(Object obj) { //利用属性来判断猫咪是否相同
    if (this == obj) { //如果两个猫咪是同一个对象则相同
        return true;
    }
    if (obj == null) { //如果两个猫咪有一个为 null 则不同
        return false;
    }
    if (getClass() != obj.getClass()) { //如果两个猫咪的类型不同则不同
        return false;
    }
    Cat cat = (Cat) obj;
    return name.equals(cat.name) && (age == cat.age)
        && (weight == cat.weight) && (color.equals(cat.color)); //比较猫咪的属性
}
@Override
public String toString() { //重写 toString()方法
    StringBuilder sb = new StringBuilder();
    sb.append("名字: " + name + "\n");
    sb.append("年龄: " + age + "\n");
    sb.append("重量: " + weight + "\n");
    sb.append("颜色: " + color + "\n");
    return sb.toString();
}
}

```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类的 main()方法中创建 3 只猫咪，并为其初始化，然后输出猫咪对象和比较的结果。关键代码如下：

```

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE); //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 3 号
        System.out.println("猫咪 1 号: " + cat1); //输出猫咪 1 号
        System.out.println("猫咪 2 号: " + cat2); //输出猫咪 2 号
        System.out.println("猫咪 3 号: " + cat3); //输出猫咪 3 号
        System.out.println("猫咪 1 号是否与猫咪 2 号相同: " + cat1.equals(cat2)); //比较是否相同
        System.out.println("猫咪 1 号是否与猫咪 3 号相同: " + cat1.equals(cat3)); //比较是否相同
    }
}

```

脚下留神：

本实例为了简单，没有重写 hashCode()方法，在实际编程中，必须同时重写该方法。



技术要点

Java 中的类都是 Object 类的直接或间接子类。在 Object 类中定义了 equals()方法用于比较类的两个对象是否相同。该方法的默认实现仅能比较两个对象是否是同一个对象。通常在定义类时推荐重写这个方法。



Note

实例 052 重新计算对象的哈希码

(实例位置: 配套资源\SL\06\052)

实例说明

Java 中创建的对象是保存在堆中的,为了提高查找的速度而使用了散列查找。散列查找的基本思想是定义一个键来映射对象所在的内存地址。当需要查找对象时,直接查找键即可,这样就不用遍历整个堆来查找对象了。本实例将查看不同对象的散列值,实例的运行效果如图 6.10 所示。

实现过程

(1) 在 Eclipse 中创建项目 052,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色,并提供构造方法来设置这些属性值。本类的重点内容在于重写 equals() 和 hashCode() 方法。重写 equals() 方法可以比较两个对象是否相同,重写 hashCode() 方法可以让相同的对象保存在相同的位置。关键代码如下:

```
public class Cat {  
    private String name; //表示猫咪的名字  
    private int age; //表示猫咪的年龄  
    private double weight; //表示猫咪的重量  
    private Color color; //表示猫咪的颜色  
    public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性  
        this.name = name;  
        this.age = age;  
        this.weight = weight;  
        this.color = color;  
    }  
    @Override  
    public boolean equals(Object obj) { //利用属性来判断猫咪是否相同  
        if (this == obj) { //如果两个猫咪是同一个对象则相同  
            return true;  
        }  
        if (obj == null) { //如果两个猫咪有一个为 null 则不同  
            return false;  
        }  
        if (getClass() != obj.getClass()) { //如果两个猫咪的类型不同则不同  
            return false;  
        }  
    }  
}
```

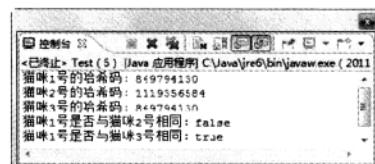


图 6.10 输出对象哈希码和比较结果



```

    }
    Cat cat = (Cat) obj;
    return name.equals(cat.name) && (age == cat.age)
        && (weight == cat.weight) && (color.equals(cat.color)); //比较猫咪的属性
}
@Override
public int hashCode() { //重写 hashCode()方法
    return 7 * name.hashCode() + 11 * new Integer(age).hashCode() + 13
        * new Double(weight).hashCode() + 17 * color.hashCode();
}
}

```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类的 main()方法中创建 3 只猫咪，并为其初始化，然后输出猫咪的哈希码和比较的结果。关键代码如下：

```

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE); //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 3 号
        System.out.println("猫咪 1 号的哈希码：" + cat1.hashCode()); //输出猫咪 1 号的哈希码
        System.out.println("猫咪 2 号的哈希码：" + cat2.hashCode()); //输出猫咪 2 号的哈希码
        System.out.println("猫咪 3 号的哈希码：" + cat3.hashCode()); //输出猫咪 3 号的哈希码
        System.out.println("猫咪 1 号是否与猫咪 2 号相同：" + cat1.equals(cat2)); //比较是否相同
        System.out.println("猫咪 1 号是否与猫咪 3 号相同：" + cat1.equals(cat3)); //比较是否相同
    }
}

```

**Note**

技术要点

一种简单的计算哈希码的方式是将重写 equals() 方法时使用到的成员变量，乘以不同的质数然后求和，以此作为新的哈希码。

实例 053 使用字符串输出对象

(实例位置：配套资源\SL\06\053)

实例说明

在日常开发中，经常需要输出对象的描述信息，此时可以通过重写 toString() 方法来完成。本实例将演示其实现过程，其运行效果如图 6.11 所示。

实现过程

(1) 在 Eclipse 中创建项目 053，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色，并提供构造方法来



图 6.11 输出对象信息



设置这些属性值。本类的重点内容在于重写 `toString()`方法，这样可以在输出对象时提供有意义的信息。关键代码如下：

```
public class Cat {  
    private String name; //表示猫咪的名字  
    private int age; //表示猫咪的年龄  
    private double weight; //表示猫咪的重量  
    private Color color; //表示猫咪的颜色  
    public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性  
        this.name = name;  
        this.age = age;  
        this.weight = weight;  
        this.color = color;  
    }  
    @Override //重写 toString()方法  
    public String toString() {  
        StringBuilder sb = new StringBuilder();  
        sb.append("名字: " + name + "\n");  
        sb.append("年龄: " + age + "\n");  
        sb.append("重量: " + weight + "\n");  
        sb.append("颜色: " + color + "\n");  
        return sb.toString();  
    }  
}
```

(3) 在 `com.mingrisoft` 包中再创建类文件，名称为 `Test`。在该类的 `main()`方法中创建 3 只猫咪，并为其初始化，然后输出猫咪对象。关键代码如下：

```
public class Test {  
    public static void main(String[] args) {  
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 1 号  
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE); //创建猫咪 2 号  
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 3 号  
        System.out.println("猫咪 1 号: " + cat1); //输出猫咪 1 号  
        System.out.println("猫咪 2 号: " + cat2); //输出猫咪 2 号  
        System.out.println("猫咪 3 号: " + cat3); //输出猫咪 3 号  
    }  
}
```

技术要点

重写 `toString()`方法时，为了给用户提供更多的信息，通常会包括类中成员变量和成员方法的介绍等。本实例中类比较简单，没有包含方法，因此简单地返回了各个成员变量的含义。为了让 `toString()`方法可以有更好的通用性，可以使用反射来获得域和方法的信息。

实例 054 Java 对象的假克隆

(实例位置：配套资源\SL\06\054)

实例说明

对象的克隆是 Java 中的一项高级技术，它可以根据给定的对象，获得与其完全相同的另一



个对象。在克隆过程中，有些注意事项。本实例将演示常见的错误，其运行效果如图 6.12 所示。

实现过程

(1) 在 Eclipse 中创建项目 054，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Employee。在 Employee 类中，定义两个成员变量分别表示员工的名字和年龄，并提供了 get() 和 set() 方法用于获得和设置变量值。重写 toString() 方法来方便对象的输出。关键代码如下：

```
public class Employee {
    private String name; //表示员工的名字
    private int age; //表示员工的年龄
    //省略 get() 和 set() 方法的相关代码
    @Override
    public String toString() { //重写 toString() 方法
        return "姓名：" + name + ", 年龄：" + age;
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中，首先创建 employee1 对象并修改其属性，再利用 “=” 将其赋值给 employee2，最后分别输出两个对象。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("克隆之前：");
        Employee employee1 = new Employee();
        employee1.setName("张 XX"); //创建 Employee 对象 employee1
        employee1.setAge(30); //为 employee1 设置姓名
        System.out.println("员工 1 的信息："); //为 employee1 设置年龄
        System.out.println(employee1); //输出 employee1 的信息
        System.out.println("克隆之后：");
        Employee employee2 = employee1; //将 employee1 赋值给 employee2
        employee2.setName("李 XX"); //为 employee2 设置姓名
        employee2.setAge(24); //为 employee2 设置年龄
        System.out.println("员工 1 的信息："); //输出 employee1 的信息
        System.out.println(employee1);
        System.out.println("员工 2 的信息："); //输出 employee2 的信息
        System.out.println(employee2);
    }
}
```

技术要点

Java 中，对于基本类型可以使用 “=” 来进行克隆，此时两个变量除了相等是没有任何关系的。而对于引用类型却不能简单地使用 “=” 进行克隆，这与 Java 的内存空间使用有关。Java 将内存空间分成两块，即栈和堆。在栈中保存基本类型和引用变量；在堆中保存对象。对于引用变量而言，使用 “=” 将修改引用，而不是复制堆中的对象。此时两个引用变量将指向同一个对象。因此，如果一个变量对其进行修改则会改变另一个变量。

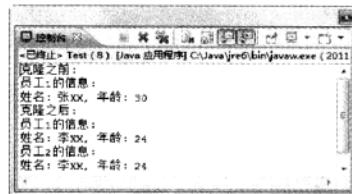


图 6.12 输出克隆前后对象信息



Note



实例 055 Java 对象的浅克隆

(实例位置: 配套资源\SL\06\055)



实例说明

Note

在克隆对象时,如果对象的成员变量是基本类型,则使用浅克隆即可完成。如果对象的成员变量包括可变引用类型,则需要使用深克隆。本实例将演示如何使用浅克隆,其运行效果如图 6.13 所示。

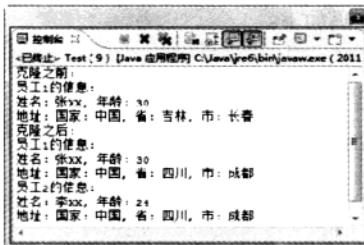


图 6.13 输出克隆前后对象信息

多学两招:

如果引用类型是不可变的,如 String 类的对象,则不必进行深克隆。

实现过程

(1) 在 Eclipse 中创建项目 055,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Address。在 Address 类中定义 3 个成员变量,分别表示地址中的国家、省和市,使用构造方法可以对它们进行赋值。此外,可以使用 get() 和 set() 方法来获取和修改成员变量的值,最后重写 toString() 方法。关键代码如下:

```
public class Address {  
    private String state; // 表示员工所在的国家  
    private String province; // 表示员工所在的省  
    private String city; // 表示员工所在的市  
    public Address(String state, String province, String city) { // 利用构造方法进行初始化  
        this.state = state;  
        this.province = province;  
        this.city = city;  
    }  
    //省略 get() 和 set() 方法  
    @Override  
    public String toString() { // 重写 toString() 方法  
        StringBuilder sb = new StringBuilder();  
        sb.append("国家: " + state + ", ");  
        sb.append("省: " + province + ", ");  
        sb.append("市: " + city);  
        return sb.toString();  
    }  
}
```



(3) 在 com.mingrisoft 包中再创建类文件，名称为 Employee。在 Employee 中定义 3 个成员变量，分别表示员工的姓名、年龄和地址。使用构造方法可以对它们进行赋值。此外，可以使用 get() 和 set() 方法来获取和修改成员变量的值，最后重写 toString() 和 clone() 方法。关键代码如下：

```

public class Employee implements Cloneable {
    private String name; //表示员工的姓名
    private int age; //表示员工的年龄
    private Address address; //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法进行初始化
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() { //重写 toString() 方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名: " + name + ", ");
        sb.append("年龄: " + age + "\n");
        sb.append("地址: " + address);
        return sb.toString();
    }
    @Override
    public Employee clone() { //实现浅克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return employee;
    }
}

```

(4) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中，首先创建 address 对象并对其初始化，然后创建 employee1 对象并对其初始化。使用 employee1 的克隆方法创建 employee2 对象。修改 employee2 的 address 属性，最后将两个对象输出。关键代码如下：

```

public class Test {
    public static void main(String[] args) {
        System.out.println("克隆之前: ");
        Address address = new Address("中国", "吉林", "长春"); //创建 address 对象
        Employee employee1 = new Employee("张 XX", 30, address); //创建 employee1 对象
        System.out.println("员工 1 的信息: ");
        System.out.println(employee1); //输出 employee1 对象
        System.out.println("克隆之后: ");
        Employee employee2 = employee1.clone(); //使用克隆创建 employee2 对象
        employee2.getAddress().setState("中国"); //修改员工地址
        employee2.getAddress().setProvince("四川"); //修改员工地址
        employee2.getAddress().setCity("成都"); //修改员工地址
        employee2.setName("李 XX"); //修改员工名字
    }
}

```



Note



```

        employee2.setAge(24);                                //修改员工年龄
        System.out.println("员工 1 的信息: ");
        System.out.println(employee1);                        //输出 employee1 对象
        System.out.println("员工 2 的信息: ");
        System.out.println(employee2);                        //输出 employee2 对象
    }
}

```



技术要点

当需要克隆对象时，需要使用 `clone()`方法，该方法的声明如下：

```
protected Object clone() throws CloneNotSupportedException
```

需要注意的是，该方法是一个受保护的方法，通常需要重写该方法并将访问权限限定符改成 `public`。该方法将类中各个域进行复制，如果对于引用类型的域，这种操作就会有问题，因此称作浅克隆。提供克隆功能的类需要实现 `Cloneable` 接口，否则使用 `clone()`方法时会抛出 `CloneNotSupportedException` 异常。

实例 056 Java 对象的深克隆

(实例位置：配套资源\SL\06\056)

实例说明

如果类的成员变量中包括可变引用类型，则在克隆时就需要使用深克隆。本实例将演示如何使用深克隆，其运行效果如图 6.14 所示。

实现过程

(1) 在 Eclipse 中创建项目 056，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `Address`。在 `Address` 类中定义 3 个成员变量，分别表示地址中的国家、省和市，使用构造方法可以对它们进行赋值。此外，可以使用 `get()` 和 `set()` 方法来获取和修改成员变量的值，最后重写 `toString()` 和 `clone()` 方法。关键代码如下：

```

public class Address implements Cloneable {
    private String state;                                //表示员工所在的国家
    private String province;                            //表示员工所在的省
    private String city;                               //表示员工所在的市
    public Address(String state, String province, String city) { //利用构造方法进行初始化
        this.state = state;
        this.province = province;
        this.city = city;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() {                           //重写 toString() 方法
        StringBuilder sb = new StringBuilder();

```

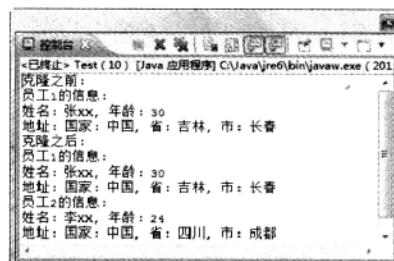


图 6.14 输出克隆前后对象信息



```

        sb.append("国家: " + state + ", ");
        sb.append("省: " + province + ", ");
        sb.append("市: " + city);
        return sb.toString();
    }
    @Override
    protected Address clone() { //实现浅克隆
        Address address = null;
        try {
            address = (Address) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return address;
    }
}

```

脚下留神：

Address 类的域不是基本类型就是不可变类型，所以可以直接使用浅克隆。

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Employee。在 Employee 中定义 3 个成员变量，分别表示员工的姓名、年龄和地址，使用构造方法可以对它们进行赋值。此外，可以使用 get() 和 set() 方法来获取和修改成员变量的值，最后重写 toString() 和 clone() 方法。关键代码如下：

```

public class Employee implements Cloneable {
    private String name; //表示员工的姓名
    private int age; //表示员工的年龄
    private Address address; //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法进行初始化
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() { //重写 toString() 方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名: " + name + ", ");
        sb.append("年龄: " + age + "\n");
        sb.append("地址: " + address);
        return sb.toString();
    }
    @Override
    public Employee clone() { //实现浅克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
            employee.address = address.clone();
        }
    }
}

```



Note



```
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return employee;
    }
}
```



Note

多学两招：

在 Java 5.0 版中，支持重写方法时返回协变类型，因此可以返回 Employee 对象。

(4) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中，首先创建 address 对象并对其初始化，然后创建 employee1 对象并对其初始化。使用 employee1 的克隆方法创建 employee2 对象，修改 employee2 的 address 属性，最后将两个对象输出。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("克隆之前：");
        Address address = new Address("中国", "吉林", "长春"); // 创建 address 对象
        Employee employee1 = new Employee("张 XX", 30, address); // 创建 employee1 对象
        System.out.println("员工 1 的信息：");
        System.out.println(employee1); // 输出 employee1 对象
        System.out.println("克隆之后：");
        Employee employee2 = employee1.clone(); // 使用克隆创建 employee2 对象
        employee2.getAddress().setState("中国"); // 修改员工地址
        employee2.getAddress().setProvince("四川"); // 修改员工地址
        employee2.getAddress().setCity("成都"); // 修改员工地址
        employee2.setName("李 XX"); // 修改员工名字
        employee2.setAge(24); // 修改员工年龄
        System.out.println("员工 1 的信息：");
        System.out.println(employee1); // 输出 employee1 对象
        System.out.println("员工 2 的信息：");
        System.out.println(employee2); // 输出 employee2 对象
    }
}
```

技术要点

通常情况下，需要克隆对象时都需要使用深克隆。但需要注意的是，如果引用类型中还有可变的引用类型成员变量，则它也需要进行克隆。例如本实例中 Address 类如果增加了一个 Date 成员变量表示开始在此居住的时间，则其也需要被克隆。

实例 057 序列化与对象克隆

(实例位置：配套资源\SL\06\057)

实例说明

如果类的成员变量比较复杂，例如使用了多个可变引用类型，使用 clone() 方法来克隆是非



常麻烦的。此时，也可以考虑使用序列化的方式完成克隆。本实例将演示其用法，实例的运行效果如图 6.15 所示。

实现过程

(1) 在 Eclipse 中创建项目 057，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Address。在 Address 类中定义 3 个成员变量，分别表示地址中的国家、省和市，使用构造方法可以对它们进行赋值。此外，可以使用 get() 和 set() 方法来获取和修改成员变量的值，最后重写 toString() 方法。关键代码如下：

```
public class Address implements Serializable {
    private static final long serialVersionUID = 4983187287403615604L;
    private String state; //表示员工所在的国家
    private String province; //表示员工所在的省
    private String city; //表示员工所在的市
    public Address(String state, String province, String city) { //利用构造方法初始化各个域
        this.state = state;
        this.province = province;
        this.city = city;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() { //使用地址属性表示地址对象
        StringBuilder sb = new StringBuilder();
        sb.append("国家: " + state + ",");
        sb.append("省: " + province + ",");
        sb.append("市: " + city);
        return sb.toString();
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Employee。在 Employee 中定义 3 个成员变量，分别表示员工的姓名、年龄和地址，使用构造方法可以对它们进行赋值。此外，可以使用 get() 和 set() 方法来获取和修改成员变量的值，最后重写 toString() 方法。关键代码如下：

```
public class Employee implements Serializable {
    private static final long serialVersionUID = 3049633059823371192L;
    private String name; //表示员工的姓名
    private int age; //表示员工的年龄
    private Address address; //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法初始化各个域
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() { //重写 toString() 方法
    }
}
```

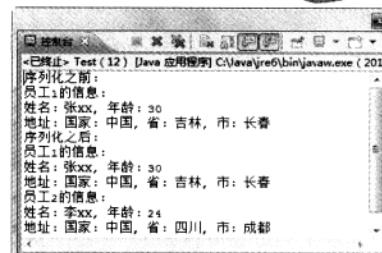


图 6.15 输出序列化前后对象信息



Note



```
StringBuilder sb = new StringBuilder();
sb.append("姓名: " + name + ",");
sb.append("年龄: " + age + "\n");
sb.append("地址: " + address);
return sb.toString();
}
```

{



Note

(4) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中，首先创建 address 对象并对其初始化，然后创建 employee1 对象并对其初始化。接着将 employee1 对象写入到本地文件 employee.dat，然后再读取出来赋值给 employee2。接着修改 employee2 的 address 属性，最后将两个对象输出。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("序列化之前: ");
        Address address = new Address("中国", "吉林", "长春"); // 创建 address 对象
        Employee employee1 = new Employee("张 XX", 30, address); // 创建 employee1 对象
        System.out.println("员工 1 的信息: ");
        System.out.println(employee1); // 输出 employee1 对象
        System.out.println("序列化之后: ");
        ObjectOutputStream out = null;
        ObjectInputStream in = null;
        Employee employee2 = null;
        try {
            out = new ObjectOutputStream(new FileOutputStream("employee.dat"));
            out.writeObject(employee1); // 将对象写入到本地文件中
            in = new ObjectInputStream(new FileInputStream("employee.dat"));
            employee2 = (Employee) in.readObject(); // 从本地文件读取对象
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            // 省略释放资源代码
        }
        if (employee2 != null) {
            employee2.getAddress().setState("中国"); // 修改员工地址
            employee2.getAddress().setProvince("四川"); // 修改员工地址
            employee2.getAddress().setCity("成都"); // 修改员工地址
            employee2.setName("李 XX"); // 修改员工名字
            employee2.setAge(24); // 修改员工年龄
            System.out.println("员工 1 的信息: ");
            System.out.println(employee1); // 输出 employee1 对象
            System.out.println("员工 2 的信息: ");
            System.out.println(employee2); // 输出 employee2 对象
        }
    }
}
```



技术要点

进行序列化的类需要实现 `Serializable` 接口，该接口中并没有定义任何方法，是一个标识接口。如果类中有可变的引用类型成员变量，则该变量也需要实现 `Serializable` 接口，依此类推。本实例中采用将对象写入本地文件的方式完成序列化，读者也可以将其写入到内存中再读取。

**Note**

实例 058 深克隆效率的比较

(实例位置：配套资源\SL\06\058)

实例说明

前面介绍了两种实现深克隆的方式，本实例将分别使用这两种方式克隆 100000 个对象，并输出花费的时间，这样读者可以对它们的效率差异有个直观的认识。实例的运行效果如图 6.16 所示。

实现过程

- (1) 在 Eclipse 中创建项目 058，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建类文件，名称为 `Employee`。在 `Employee` 类中定义两个成员变量，分别表示员工的姓名和年龄，并重写 `toString()` 和 `clone()` 方法。关键代码如下：

```
public class Employee implements Cloneable, Serializable {
    private static final long serialVersionUID = 5022956767440380940L;
    private String name; //表示员工的姓名
    private int age; //表示员工的年龄
    public Employee(String name, int age) { //利用构造方法初始化各个域
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString() { //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名：" + name + ", ");
        sb.append("年龄：" + age + "\n");
        return sb.toString();
    }
    @Override
    protected Employee clone() { //使用父类的 clone()方法实现深克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return employee;
    }
}
```

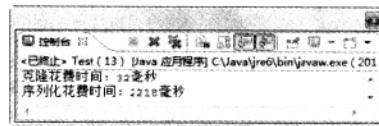


图 6.16 克隆与序列化花费的时间



(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中先创建一个列表保存 Employee 对象，然后分别使用克隆和序列化的方式依次获得 10000 个对象，并计算输出花费的时间。关键代码如下：



Note

```
public class Test {  
    public static void main(String[] args) {  
        List<Employee> employees = new ArrayList<Employee>(); // 创建列表保存对象  
        Employee employee = new Employee("李 XX", 25); // 创建 Employee 类的对象  
        long currentTime = System.currentTimeMillis(); // 获得当前系统时间  
        for (int i = 0; i < 100000; i++) {  
            employees.add(employee.clone()); // 使用克隆方式获得对象  
        }  
        System.out.println("克隆花费时间：" + (System.currentTimeMillis() - currentTime) + "毫秒");  
        currentTime = System.currentTimeMillis(); // 获得当前时间  
        for (int i = 0; i < 100000; i++) {  
            ByteArrayOutputStream baos = new ByteArrayOutputStream(); // 创建字节数组输出流  
            ObjectOutputStream out = null;  
            try {  
                out = new ObjectOutputStream(baos); // 创建对象输出流  
                out.writeObject(employee); // 将对象写入到输出流中  
            } catch (IOException e) {  
                e.printStackTrace();  
            } finally {  
                // 省略释放资源代码  
            }  
            // 获得字节数组输出流内容  
            ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());  
            ObjectInputStream in = null;  
            try {  
                in = new ObjectInputStream(bais); // 创建对象输入流  
                employees.add((Employee) in.readObject()); // 读取对象  
            } catch (IOException e) {  
                e.printStackTrace();  
            } catch (ClassNotFoundException e) {  
                e.printStackTrace();  
            } finally {  
                // 省略释放资源代码  
            }  
        }  
        System.out.println("序列化花费时间：" + (System.currentTimeMillis() - currentTime) + "毫秒");  
    }  
}
```

技术要点

使用 ByteArrayOutputStream 和 ByteArrayInputStream 可以将对象保存在内存中，这样就不必产生一个本地文件来完成序列化的功能。