

第 8 章

字符串与包装类

本章读者可以学到如下实例：

- ▶ 实例 079 将数字格式化为货币字符串
- ▶ 实例 080 货币金额大写格式
- ▶ 实例 081 String 类格式化当前日期
- ▶ 实例 082 字符串大小写转换
- ▶ 实例 083 字符与 Unicode 码的转换
- ▶ 实例 084 判断用户名是否正确
- ▶ 实例 085 用户名排序
- ▶ 实例 086 判断网页请求与 FTP 请求
- ▶ 实例 087 判断文件类型
- ▶ 实例 088 判断字符串是否为数字
- ▶ 实例 089 验证 IP 地址的有效性
- ▶ 实例 090 鉴别非法电话号码
- ▶ 实例 091 将字符串转换成整数
- ▶ 实例 092 整数进制转换器
- ▶ 实例 093 获取字符串中汉字的个数
- ▶ 实例 094 批量替换某一类字符串
- ▶ 实例 095 查看数字的取值范围
- ▶ 实例 096 ASCII 编码查看器
- ▶ 实例 097 判断手机号的合法性
- ▶ 实例 098 用字符串构建器追加字符
- ▶ 实例 099 去掉字符串中的所有空格
- ▶ 实例 100 Double 类型的比较



实例 079 将数字格式化为货币字符串

(实例位置: 配套资源\SL\08\079)



Note

数字可以标识货币、百分比、积分、电话号码等,就货币而言,在不同的国家会以不同的格式来定义。本实例将接收用户输入的数字,将这个数字转换为不同国家的货币格式,然后在控制台中输出这些货币格式。实例的运行效果如图 8.1 所示。

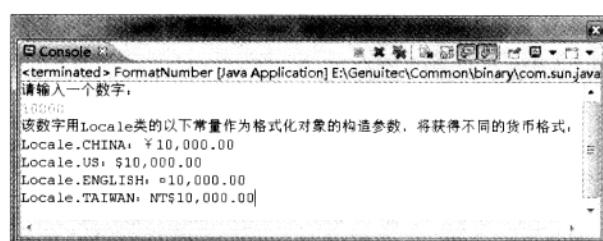


图 8.1 将数字格式化为货币字符串

实现过程

本实例首先创建 FormatNumber 类,在该类的主方法中创建标准输入流的扫描器对象,利用该对象从键盘上接收一个需要转换为货币格式的数字,通过 NumberFormat 类的 format()方法把接收的数字格式转换为货币字符串。关键代码如下:

```
public class FormatNumber {  
    public static void main(String[] args) {  
        Scanner scan = new Scanner(System.in); // 创建标注输入流扫描器  
        System.out.println("请输入一个数字: "); // 获取用户输入  
        double number = scan.nextDouble(); // 获取用户输入数字  
        System.out.println("该数字用 Locale 类的常量作为格式化对象的构造参数, 将获得不同的货币格式: ");  
        NumberFormat format = NumberFormat.getCurrencyInstance(Locale.CHINA); // 创建格式化对象  
        System.out.println("Locale.CHINA: " + format.format(number)); // 输出格式化货币格式  
        format = NumberFormat.getCurrencyInstance(Locale.US);  
        System.out.println("Locale.US: " + format.format(number));  
        format = NumberFormat.getCurrencyInstance(Locale.ENGLISH);  
        System.out.println("Locale.ENGLISH: " + format.format(number));  
        format = NumberFormat.getCurrencyInstance(Locale.TAIWAN);  
        System.out.println("Locale.TAIWAN: " + format.format(number));  
    }  
}
```

技术要点

数字格式化是本实例的关键点,实例中应用 NumberFormat 类实现了数字格式化,这个类是一个抽象类,但是可以通过其静态方法获取内部实现类的实例对象,本实例获取了货币格式



的格式化对象。

1. 获取货币格式对象

使用 `getCurrencyInstance()`方法获取 `NumberFormat` 类的货币格式对象。该方法的声明如下：

```
public static NumberFormat getCurrencyInstance(Locale inLocale);
```

参数说明

`inLocale`: 指定语言环境。

返回值：返回指定语言环境的货币格式。

2. 执行格式化

`format()`方法是格式化对象中的方法，用于执行针对数字的格式化操作，就本实例使用的货币格式化对象来说，这个方法执行的是将数字格式化为货币字符串。该方法的声明如下：

```
public final String format(double number);
```

参数说明

`number`: 要被格式化的数字。

多学两招：

格式化对象可以指定语言环境，在 Java 中使用 `Local` 类的对象来表示，在该类中包含了各种语言环境。通过它可以获取国际化的字符串信息，如货币、日期时间等。



Note

实例 080 货币金额大写格式

(实例位置：配套资源\SL\08\080)

实例说明

在处理财务账款时，一般需要使用大写金额。如进行转账时，需要将转账金额写成大写的。也就是说，如果要转账 123456.00 元，则需要写成“壹拾贰万叁仟肆佰伍拾陆元整”。对于这种情况，如果手动填写不仅麻烦，而且容易出错，所以常常需要通过程序控制自动进行转换。本实例实现了小写金额到大写金额的转换，实例的运行效果如图 8.2 所示。

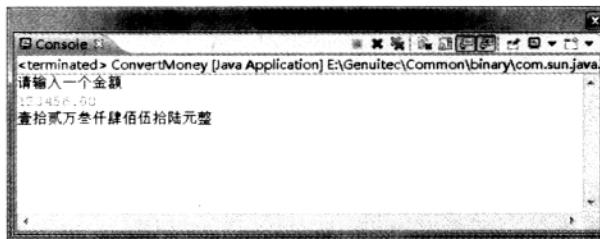


图 8.2 将金额转换为大写格式

实现过程

(1) 在项目中创建 `ConvertMoney` 类，在该类的主方法中接收用户输入的金额，然后通过 `convert()`方法把这个金额转换成大写金额的字符串格式，并输出到控制台。



```
public static void main(String[] args) {  
    Scanner scan = new Scanner(System.in); //创建扫描器  
    System.out.println("请输入一个金额");  
    String convert = convert(scan.nextDouble()); //获取金额转换后的字符串  
    System.out.println(convert); //输出转换结果  
}
```



Note

(2) 编写金额转换方法 convert(), 该方法在主方法中被调用, 用于金额大写格式的转换。在该方法中创建 DecimalFormat 类的实例对象, 通过这个格式器对象把金额数字格式化, 值保留 3 位小数。然后分别调用 getInteger()与 getDecimal()方法转换整数与小数部分, 并返回转换后的结果。关键代码如下:

```
public static String convert(double d) {  
    DecimalFormat df = new DecimalFormat("#0.###"); //实例化 DecimalFormat 对象  
    String strNum = df.format(d); //格式化 double 数字  
    if (strNum.indexOf(".") != -1) { //判断是否包含小数点  
        String num = strNum.substring(0, strNum.indexOf("."));  
        if (num.length() > 12) { //整数部分大于 12 不能转换  
            System.out.println("数字太大, 不能完成转换! ");  
            return "";  
        }  
        String point = ""; //小数点  
        if (strNum.indexOf(".") != -1) {  
            point = "元";  
        } else {  
            point = "元整";  
        }  
        String result = getInteger(strNum) + point + getDecimal(strNum); //转换结果  
        if (result.startsWith("元")) { //判断字符串是否以“元”结尾  
            result = result.substring(1, result.length()); //截取字符串  
        }  
        return result; //返回新的字符串  
    }  
}
```

(3) 编写 getInteger()方法, 用于转换数字整数部分的大写格式。在该方法中判断数字是否包含小数点, 然后把数字转换为字符串并反转字符顺序, 为每个数字添加对应的大写单位。关键代码如下:

```
public static String getInteger(String num) {  
    if (num.indexOf(".") != -1) { //判断是否包含小数点  
        num = num.substring(0, num.indexOf("."));  
    }  
    num = new StringBuffer(num).reverse().toString(); //反转字符串  
    StringBuffer temp = new StringBuffer(); //创建一个 StringBuffer 对象  
    for (int i = 0; i < num.length(); i++) { //加入单位  
        temp.append(STR_UNIT[i]);  
        temp.append(STR_NUMBER[num.charAt(i) - 48]);  
    }  
    num = temp.reverse().toString(); //反转字符串  
    num = numReplace(num, "零拾", "零"); //替换字符串的字符  
    num = numReplace(num, "零佰", "零"); //替换字符串的字符
```



```

num = numReplace(num, "零仟", "零");           //替换字符串的字符
num = numReplace(num, "零万", "万");           //替换字符串的字符
num = numReplace(num, "零亿", "亿");           //替换字符串的字符
num = numReplace(num, "零零", "零");           //替换字符串的字符
num = numReplace(num, "亿万", "亿");           //替换字符串的字符
if (num.lastIndexOf("零") == num.length() - 1) { //如果字符串以零结尾将其除去
    num = num.substring(0, num.length() - 1);
}
return num;
}

```

技术要点

实现本实例关键在于以下几点：

- 将数字格式化，如果存在小数部分，将其转换为3位小数，精确到厘。
- 分别将整数部分与小数部分转换为大写方式，并插入其单位（亿、万、仟……）。
- 组合转换后的整数部分与小数部分。

多学两招：

DecimalFormat 类可以指定格式化模板来格式化浮点数，如保留几位小数。通过调用该类的 format()方法可以使用指定模板来格式化任意浮点数字。

实例 081 String 类格式化当前日期

(实例位置：配套资源\SL\08\081)

实例说明

在输出日期信息时，经常需要输出不同格式的日期格式，本实例中介绍了 String 字符串类中的日期格式化方法，实例使用不同的方式输出 String 类的日期格式参数值，组合这些值可以实现特殊格式的日期字符串。实例的运行效果如图 8.3 所示。

实现过程

(1) 在项目中创建 Example 类，在该类的主方法中创建一个 Date 日期类的实例对象 today。

(2) 利用 format()方法格式化 today 日期对象，关键代码如下：

```

public class Example {
    public static void main(String[] args) {
        Date today = new Date();
        String a = String.format(Locale.US, "%tb", today); //格式化后的字符串为月份的英文缩写
        System.out.println("格式化后的字符串为月份的英文缩写：" + a);
        String b = String.format(Locale.US, "%tB", today); //格式化后的字符串为月份的英文全写
        System.out.println("格式化后的字符串为月份的英文全写：" + b);
    }
}

```

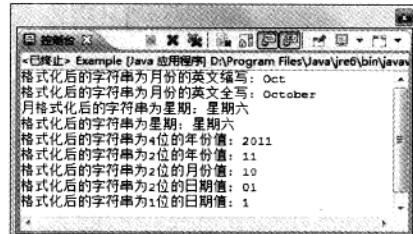


图 8.3 格式化当前日期



Note



```
String c = String.format("%ta", today);           //格式化后的字符串为星期（如星期一）
System.out.println("月格式化后的字符串为星期: " + c);
String d = String.format("%tA", today);           //格式化后的字符串为星期（如星期二）
System.out.println("格式化后的字符串为星期: " + d);
String e = String.format("%tY", today);           //格式化后的字符串为 4 位的年份值
System.out.println("格式化后的字符串为 4 位的年份值: " + e);
String f = String.format("%ty", today);           //格式化后的字符串为 2 位的年份值
System.out.println("格式化后的字符串为 2 位的年份值: " + f);
String g = String.format("%tm", today);           //格式化后的字符串为 2 位的月份值
System.out.println("格式化后的字符串为 2 位的月份值: " + g);
String h = String.format("%td", today);           //格式化后的字符串为 2 位的日期值
System.out.println("格式化后的字符串为 2 位的日期值: " + h);
String i = String.format("%te", today);           //格式化后的字符串为 1 位的日期值
System.out.println("格式化后的字符串为 1 位的日期值: " + i);
}
}
```

技术要点

使用 String 类的 format()方法不但可完成日期的格式化，也可实现时间的格式化。时间格式化转换符要比日期转换符更多、更精确，它可以将时间格式化为时、分、秒、毫秒。

多学两招：

在深入使用字符串之前，有一个概念一定要理解，字符串是不可变的对象。理解了这一概念，对后面熟练使用字符串有着很大的帮助。字符串的不可变性，意味着每当对字符串进行操作时，都将产生一个新的字符串对象，如果频繁地操作字符串对象，会在托管堆中产生大量无用字符串，增加垃圾收集器的压力，从而造成系统资源的浪费。

实例 082 字符串大小写转换

(实例位置：配套资源\SL\08\082)

实例说明

在程序设计过程中，经常会遇到一种情况，在验证用户登录时，如果用户名不区分大小写，那么在代码中应当使用一种方法排除字母大小写的因素，然后再对比数据库中的用户名与用户输入的用户名是否相等。可以先将数据库中的用户名全部转为大写，再将用户输入的用户名转为大写，最后对比是否相等。本实例中所介绍的技术可以方便地将字符串中的字母全部转换为大写或小写。实例的运行效果如图 8.4 所示。

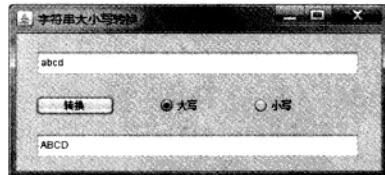


图 8.4 大小写转换

实现过程

(1) 创建窗体，在窗体中放置两个文本框，一个用于输入字符串，另一个用于显示结果，



添加两个单选按钮和一个“转换”按钮控件。

(2) 编写“转换”按钮的事件处理方法，在该方法中获取用户输入的字符串，根据用户的选择进行大小写格式转换并把结果输出到文本框中。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent arg0) {
    String command = buttonGroup.getSelection().getActionCommand();
    //获取“大写”和“小写”单选按钮的选中
    boolean upper = command.equals("大写");
    //判断是否选中“大写”单选按钮
    String text = inputTextField.getText();
    //获取输入字符串
    if (upper) {
        outputTextField.setText(text.toUpperCase());
        //大写转换
    } else {
        outputTextField.setText(text.toLowerCase());
        //小写转换
    }
}
```



Note

指点迷津：

字符串在创建后就成为不可变的对象，当调用字符串对象的方法操作字符串时，会产生新的字符串对象，而不是更改原来的字符串对象。

技术要点

本实例实现时主要用到了字符串对象的 `toUpperCase()` 和 `toLowerCase()` 方法，下面对其进 行详细讲解。

使用字符串对象的 `toUpperCase()` 方法可以将字符串中的字母全部转换为大写。格式如图 8.5 所示。

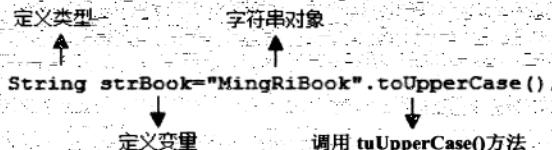


图 8.5 调用字符串对象的 `toUpperCase()` 方法将字母全部转为大写

从图 8.5 中可以看到，字符串对象调用 `toUpperCase()` 方法后，会返回一个将原字符串转换为大写的新字符串，并将新字符串的引用交给 `strBook` 变量。

使用字符串对象的 `toLowerCase()` 方法可以将字符串中的字母全部转换为小写。格式如图 8.6 所示。



图 8.6 调用字符串对象的 `toLowerCase()` 方法将字母全部转为小写

从图 8.6 中可以看到，字符串对象调用 `toLowerCase()` 方法后，会返回一个将原字符串转换为小写的新字符串，并将新字符串的引用交给 `strBook` 变量。



多学两招：

单选按钮控件需要分组规划，例如本实例中的两个单选按钮，如果不添加到 ButtonGroup 中，就无法显现单选操作，也就是说两个单选按钮可以同时处于选中状态。



实例 083 字符与 Unicode 码的转换

(实例位置：配套资源\SL\08\083)

实例说明

Unicode 是一种字符编码，它可以显示各国语言的各种文字、标点、制表符等所有字符，也是现今最通用的字节编码系统。在程序设计中，可以方便地将字符转换为 Unicode 码，也可以将 Unicode 码转换为字符。实例的运行效果如图 8.7 所示。

实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建窗体类 CharacterASCII，在该窗体中添加 4 个文本框和两个按钮，其中文本框用于接收用户输入的字符和编码以及输出转换结果。两个按钮分别用于控制字符到 Unicode 编码的转换和 Unicode 编码到字符的转换。

(2) 编写“转换为 Unicode 码”按钮的事件处理方法，在该方法中获取用户输入的字符串，然后从该字符串中提取字符数组，在遍历该数组的同时，把每个字符的编码输出到文本框。关键代码如下：

```
protected void do_codeButtonActionPerformed(ActionEvent e) {  
    String text = charInputField.getText(); // 获取用户输入的字符串  
    char[] charArray = text.toCharArray(); // 获取字符串的字符数组  
    StringBuilder builder = new StringBuilder(); // 创建字符串构建器  
    for (char c : charArray) { // 遍历字符数组  
        builder.append((int) c + " "); // 连接各字符的编码  
    }  
    codeOutputField.setText(builder.toString()); // 将结果输出到文本框  
}
```

(3) 编写“转换为字符”按钮的事件处理方法，在该方法中获取用户输入的 Unicode 编码，然后通过强制类型转换，获取该编码对应的字符并输出到文本框中。关键代码如下：

```
protected void do_charButtonActionPerformed(ActionEvent e) {  
    Number value = (Number) codeInputField.getValue(); // 获取用户输入 Unicode 编码  
    long code = value.longValue(); // 取输入数字的 Long 类型值  
    charOutputField.setText(((char) code)+""); // 输出编码到文本框  
}
```

指点迷津：

字符串就是由多个字符组成的，灵活地运用字符数组可以实现复杂的字符串操作，通过数组下标的各种算法可以使字符串更加灵活。

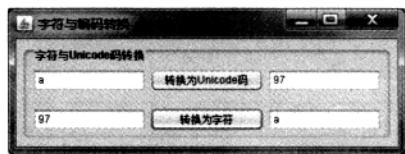


图 8.7 字符与 Unicode 码的转换



技术要点

本实例使用了字符串对象的 `toCharArray()` 方法获取字符数组，数组中的每个元素都是字符串的一部分。其声明语法如下：

```
public char[] toCharArray();
```

返回值：字符串中每个字符组成的字符数组。



Note

多学两招：

现在已经知道 `char` 是值类型，可以将字母强制类型转换为整数数值，从而方便地得到字母的 Unicode 编码。同样地，可以将整数数值强制类型转换为 `char`，从而得到对应编码的字符。

实例 084 判断用户名是否正确

(实例位置：配套资源\SL\08\084)

实例说明

在程序的开发过程中，经常需要判断用户输入的用户名是否正确，可以通过对比用户输入的用户名字符串是否与数据库中或者已经存在集合中的字符串相同来决定用户输入的用户名是否正确。Java 的基本数据类型可以使用 “`==`” 判断两个操作数是否相等，但是对于 Java 类创建的对象就不能使用这种方法来判断是否相等了。字符串是基本数据类型之外的，也就是说字符串在 Java 中是对象。本实例将通过字符串相等判断来实现用户名验证。实例的运行效果如图 8.8 所示。

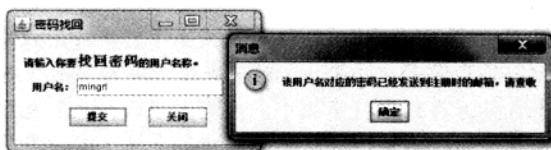


图 8.8 判断用户名是否正确

实现过程

(1) 在项目中创建窗体类，在窗体中添加接收用户输入信息的文本框、“提交”和“关闭”按钮。

(2) 编写“提交”按钮的事件处理方法，在该方法中接收用户输入的用户名，然后判断输入，如果不是管理员用户名并且输入的用户名是已经注册的则显示正确提示，否则显示错误提示。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String name = usernameField.getText(); // 获取用户输入
    if (name.equals("admin")) { // 判断是否是管理员账号
        JOptionPane.showMessageDialog(null, "对不起，这个用户名是管理员的，不是你的");
    } else if (name.equals("mingri")) { // 判断是否注册用户
        JOptionPane.showMessageDialog(null, "该用户名对应的密码已经发送到注册时的邮箱，请查收");
    } else { // 给错误用户名的提示对话框
        JOptionPane.showMessageDialog(null, "你输入的用户名不存在，留意 Caps Lock 键是否按下。");
    }
}
```



技术要点

本实例调用了 String 类的 equals()方法来判断两个字符串内容是否相同，这个方法是从 Object 类中继承的。在 Java 语言中，默认所有类都是 Object 类的子类，也就是说只要是对象，都会重写或直接使用 Object 类的 equals()方法，String 类就重写了这个方法实现判断字符串内容是否相同。其声明语法如下：

```
public boolean equals(Object anObject);
```

参数说明

anObject：与当前字符串进行比较的对象。

返回值：如果给定对象表示的 String 与此 String 相等，则返回 true；否则返回 false。

多学两招：

在 Java 虚拟机中有一个保存字符串的池，它会记录所有字符串。例如：

```
String str1="abc";  
String str2="abc";  
String str3=new String("abc");  
System.out.println(str1==str2);  
System.out.println(str1==str3);
```

这段代码中“str1==str2”的判断将返回 true，为什么这个等式会成立呢？Java 中基本数据类型使用“==”可以判断操作数是否相等，对于对象使用这个符号判断的是两个对象的内存地址是否相同。而 Java 虚拟机为了提高字符串应用效率，提供了字符串池来保存字符串常量，str1 创建字符串常量“abc”，这时会先检测字符串池中是否包含该字符串，如果不包含，则创建字符串常量保存到字符串池中，然后再返回。str2 也赋值为字符串“abc”，这是由于字符串池中已经存在该字符串，所以不再创建，直接返回该字符串，也就是说这两个变量引用同一个字符串，那么它们的内存地址也是相同的，所以 str1==str2 成立。但是使用 new 关键字创建的字符串会新开辟内存控件，所以 str1==str3 不成立。

实例 085 用户名排序

(实例位置：配套资源\SL\08\085)

实例说明

用户名也就是登录系统、网站等使用的名称，也称登录名称。一般情况下，用户名都要求使用英文、数字与符号组成，如 li_zhongwei。这些用户名一般是根据用户注册的先后来排序的，这样不利于管理员的查找，本实例实现对用户名字符串进行排序，实例的运行效果如图 8.9 所示。

实现过程

(1) 在项目中创建窗体类 UserSort，在窗体界面中添加一个 JList 列表控件和“升序”、“降序”、“关闭”3

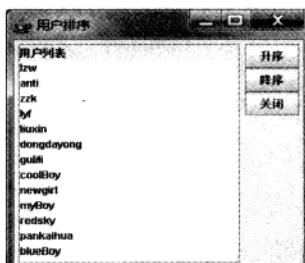


图 8.9 用户名排序



个按钮。

(2) 编写处理升序和降序按钮事件的方法，该方法将创建线程对象在新的线程中完成排序，并动态更新界面排序过程。关键代码如下：

```

protected void do_buttonActionPerformed(final ActionEvent e) {
    new Thread() {
        int[] indexs = new int[2];
        public void run() {
            for (int i = names.length; --i >= 0;) { //遍历数组
                indexs[0]=i;
                for (int j = 0; j < i; j++) { //遍历并排序所有未排序元素
                    boolean compare = names[j].compareToIgnoreCase(names[j+1]) > 0;
                    if (compare && e.getSource() == ascButton || !compare
                        && e.getSource() == descButton) { //条件判断
                        String temp = names[j]; //数组元素交换
                        names[j] = names[j+1];
                        names[j+1] = temp;
                        sourceList.repaint();
                    }
                try {
                    sleep(100);
                } catch (InterruptedException e1) {
                }
                indexs[1]=j;
                sourceList.setSelectedIndices(indexs);
            }
        }
    }.start();
    sourceList.repaint(); //更新列表控件
}
}

```



Note

技术要点

用户名是以字符串形式保存的，无论在数据库还是在数据集合中，所以本实例使用了字符串的 `compareTo()` 和 `compareToIgnoreCase()` 方法实现字符串的对比。下面介绍本实例使用的关键方法。

1. `compareTo()`方法

该方法将按字典顺序比较两个字符串。该方法比较基于字符串中各个字符的 Unicode 值。如果按字典顺序，此 `String` 对象位于参数字符串之前，则比较结果为一个负整数；如果按字典顺序，此 `String` 对象位于参数字符串之后，则比较结果为一个正整数。其声明语法如下：

```
public int compareTo(String anotherString);
```

参数说明

`anotherString`：要比较的 `String` 字符串对象。

返回值：如果参数字符串等于此字符串，则返回值 0；如果此字符串按字典顺序小于字符串参数，则返回一个小于 0 的值；如果此字符串按字典顺序大于字符串参数，则返回一个大于 0 的值。

2. `compareToIgnoreCase()`方法

同上一个方法执行的功能相同，这个方法也用于对比两个字符串，但是不再严格区分字母



的大小写。其声明语法如下：

```
public int compareIgnoreCase(String str);
```

参数说明

str：要比较的 String 字符串对象。

返回值：根据指定 String 大于、等于还是小于此 String（不考虑大小写），分别返回一个负整数、0 或一个正整数。

Note

多学两招：

本实例使用了冒泡排序算法。冒泡排序是最常用的数组排序算法之一，它排序数组元素的过程总是小数往前放，大数往后放，类似水中气泡往上升的动作，所以称作冒泡排序。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），这样较小的元素就像气泡一样从底部上升到顶部。

实例 086 判断网页请求与 FTP 请求

（实例位置：配套资源\SL\08\086）

实例说明

大家在访问 Internet 网络时，经常涉及很多访问协议，其中最明显、最常用的就是访问网页的 HTTP 协议、访问 FTP 服务器的 FTP 协议等。本实例实现对用户输入进行判断，根据用户输入的不同请求字符串，判断请求类型，如图 8.10 所示。在文本框中输入请求字符串，并单击“验证”按钮，程序将提示用户输入的请求类型。



图 8.10 验证地址

实现过程

- (1) 在项目中创建窗体类，在窗体中添加一个文本框和“验证”、“关闭”两个按钮。
- (2) 编写“验证”按钮的事件处理方法，在该方法中获取用户输入的请求字符串，然后通过 String 类的方法判断该字符串以 http 开头还是以 ftp 开头，并以对话框提示请求类型。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String request = requestField.getText(); // 获取用户输入
    if (request.startsWith("http")) { // 判断输入是否以 http 开头
        JOptionPane.showMessageDialog(null, "您输入的是网页地址，希望浏览某个网站。");
    } else if (request.startsWith("ftp")) { // 判断输入是否以 ftp 开头
        JOptionPane.showMessageDialog(null, "您输入的是 FTP 地址，希望访问 FTP 服务器。");
    } else { // 其他字符串开头认为信息不完整
    }
}
```



```
JOptionPane.showMessageDialog(null, "您输入的请求信息不完整。");
}
}
```

技术要点

本实例通过调用 String 类的 `startsWith()`方法判断字符串的前缀，根据前缀来辨别请求的类型，该方法将判断字符串是否以指定的前缀开始。其声明语法如下：

```
public boolean startsWith(String prefix);
```

参数说明

`prefix`: 字符串前缀。

返回值：如果参数表示的字符序列是此字符串表示的字符序列的前缀，则返回 `true`；否则返回 `false`。

多学两招：

Java 中一句相连的字符串不能分开在两行中写。例如：

```
System.out.println("I like
Java")
```

修改为：这种写法是错误的，无法通过编译。如果一个字符串太长，为了便于阅读，可以将这个字符串分在两行上书写。此时就可以使用“+”将两个字符串连起来，之后在加号处换行。因此上面的语句可以修改为：

```
System.out.println("I like"+
"Java");
```



Note

实例 087 判断文件类型

(实例位置：配套资源\SL\08\087)

实例说明

在计算机中使用多种类型的文件来保存不同的数据，操作员和系统程序都根据不同的类型查找相应的数据。区分不同文件类型的依据就是文件的扩展名称。本实例利用字符串的判断方法来检测文件结尾的字符串后缀，并提示用户不同文件类型的说明信息，如图 8.11 所示。

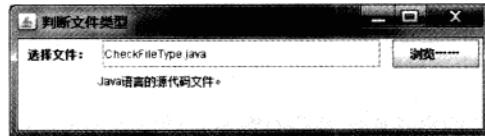


图 8.11 判断文件类型

实现过程

(1) 在项目中创建窗体类 `CheckFileType`，在该窗体中添加一个文本框和一个“浏览”按钮，另外在“浏览”按钮和文本框下方还要添加一个 `JTextArea` 文本域控件，该控件用于显示用户选择文件类型的描述信息。



(2) 编写“浏览”按钮的事件处理方法，在该方法中创建说明文件的扫描器，这个说明文件是与程序存放在一起的 extName.inf 文件，笔者在该文件中添加了部分文件类型的描述信息。接着创建文件选择器，用户可以通过该选择器选择文件，然后判断文件扩展名称并从说明文件中提取对应的说明信息显示到文本域控件中。关键代码如下：

```

protected void do_buttonActionPerformed(ActionEvent e) {
    Scanner scan = new Scanner(getClass()
        .getResourceAsStream("extName.inf")); //获取说明文件的扫描器
    JFileChooser chooser = new JFileChooser(); //创建文件选择器
    boolean searched = false;
    int option = chooser.showOpenDialog(this); //打开文件选择对话框
    if (option == JFileChooser.APPROVE_OPTION) { //如果正确选择文件
        File file = chooser.getSelectedFile(); //获取用户选择文件
        textField.setText(file.getName()); //把文件名添加到文本框
        String name = file.getName(); //获取文件名
        while (scan.hasNextLine()) { //遍历说明文件
            String line = scan.nextLine(); //获取一行说明信息
            String[] extInfo = line.split("\t"); //把单行说明信息拆分成数组
            if (name.endsWith(extInfo[0])) { //数组第一个元素是文件扩展名，与用户选择文件
                name对比
                textField.setText(extInfo[1]); //第二个数组元素是文件类型的说明信息，添
                加到文本域控件中
                searched = true;
            }
        }
        scan.close(); //关闭扫描器
    }
    if (!searched) { //如果没找到相关文件类型的说明，则提示用户
        textArea.setText("你选择的文件类型没有相应记录，你可以在 extName.info 文件中添加
该类型的描述。");
    }
}

```

技术要点

本实例使用了 String 字符串类的 `endsWith()` 方法来判断字符串结尾的后缀。该方法判断字符串是否以指定的后缀结尾。对于文件来说，结尾的后缀是文件扩展名，通过这个扩展名就可以判断文件类型，所以 `endsWith()` 方法最适合不过。其声明语法如下：

```
public boolean endsWith(String suffix);
```

参数说明

`suffix`: 后缀字符串。

返回值：如果参数表示的字符序列是此对象表示的字符序列的后缀，则返回 `true`；否则返回 `false`。

多学两招：

这个方法接收一个正则表达式字符串作为参数，通过这个表达式指定的用以分隔符来分割字符串为指定长度的字符串数组，但是如果被分割的字符串没有统一的分隔符，可以使用 “|” 定义多个分隔符。例如 “|,-!” 分别以 “,”、 “-” 和 “!” 作为分隔符。





实例 088 判断字符串是否为数字

(实例位置: 配套资源\SL\08\088)

实例说明

软件运行过程中经常需要用户输入数值、货币值等信息,然后进行处理。由于用户输入只能是字符串类型,如果输入了非法的信息,如在货币值中输入了字母“a”以及其他非数字字符,那么在运行时会抛出异常,可以通过捕获异常来判断输入信息是否合法,但是,这样并不是好的处理方法,本实例将使用 NumberUtils 类中的方法处理此问题,让程序更加方便快捷。实例的运行效果如图 8.12 所示。



图 8.12 判断输入的是不是数字

实现过程

- (1) 在项目中创建窗体类 CheckNumber, 在窗体中添加一个文本框和一个“判断”按钮。
- (2) 编写“判断”按钮的事件处理方法, 在该方法中获取用户在文本框中输入的数字, 然后利用 NumberUtils 类的 isNumber()方法判断字符串是不是有效的数字, 然后用对话框输出正确和错误的提示信息。关键代码如下:

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String text = textField.getText(); // 获取用户输入的金额字符串
    boolean isnum = NumberUtils.isNumber(text); // 判断是不是数字
    if(isnum){ // 输出正确提示信息
        JOptionPane.showMessageDialog(null, "输入正确, 是数字格式");
    }else{ // 输出错误提示信息
        JOptionPane.showMessageDialog(null, "输入错误, 请确认格式再输入");
    }
}
```

技术要点

本实例使用了 Apache 提供的 lang 包中的 NumberUtils 类来实现数字判断, 该类的全路径为 “org.apache.commons.lang.math.NumberUtils”, 这个类中的 isNumber()方法可以接收字符串参数, 然后对字符串进行解析, 如果字符串不能转换为数字格式, 则返回 false。其声明语法如下:

```
public static boolean isNumber(String str);
```

参数说明

str: 字符串。

返回值: 该方法将对该字符串进行判断, 如果是由数字组成则返回 true; 如果无法转换为数字, 则返回 false。



Note



多学两招：

本实例还可以通过 Double 类的 `parseDouble()` 方法把字符串转换为 `double` 类型，如果抛出异常说明字符串不是合法数字格式。但是建议不要使用这种方式作判断，那会降低程序性能，因为它无法与简单逻辑判断相比，后者在速度上完全超越前者。



Note

实例 089 验证 IP 地址的有效性

(实例位置：配套资源\SL\08\089)

实例说明

IP 地址是网络上每台计算机的标识，在浏览器中输入的网址也是要经过 DNS 服务器转换为 IP 地址才能找到服务器的。在很多网络程序中要求设置服务器 IP 地址或者输入对方连接 IP 地址，IP 地址的错误输入将使程序无法运行。本实例实现对 IP 地址的验证功能，实例的运行效果如图 8.13 所示，把该功能加载到网络程序中，可以避免用户输入错误的 IP 地址。

实现过程

(1) 在项目中创建窗体类 `CheckIPAddress`，在该窗体中添加一个输入 IP 地址的文本框和一个“验证”按钮。

(2) 编写“验证”按钮的事件处理方法，该方法将获取用户输入，然后调用 `matches()` 方法对输入进行判断，然后在对话框中输出结果。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String text = ipField.getText(); // 获取用户输入
    String info = matches(text); // 对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info); // 用对话框输出验证结果
}
```

(3) 编写验证 IP 地址的 `matches()` 方法，该方法利用正则表达式对输入字符串进行验证，并返回验证结果。关键代码如下：

```
public String matches(String text) {
    if(text != null && !text.isEmpty()){
        String regex = "^(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d|[1-9]\\d{2}|" +
                      "(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d\\d|" +
                      "(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d\\d\\d|" +
                      "(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d\\d\\d$"; // 定义正则表达式
        if(text.matches(regex)){
            return text + "\n 是一个合法的 IP 地址！"; // 判断 IP 地址是否与正则表达式匹配
        } else{
            return text + "\n 不是一个合法的 IP 地址！"; // 返回判断信息
        }
    }
    return "请输入要验证的 IP 地址！"; // 返回判断信息
}
```



图 8.13 验证一个 IP 地址



技术要点

本实例的关键点在于 IP 地址格式与数字范围的验证，用户在输入 IP 地址时，程序可以获取的只有字符串类型，所以本实例利用字符串的灵活性与正则表达式搭配进行 IP 格式与范围的验证。该方法是 String 字符串类的方法，用于判断字符串与指定的正则表达式是否匹配。其声明语法如下：

```
public boolean matches(String regex);
```

参数说明

regex：用来匹配此字符串的正则表达式。

返回值：当且仅当此字符串符合给定的正则表达式条件时，返回 true。



Note

多学两招：

在正则表达式中，“.” 代表任何一个字符，因此在正则表达式中如果想使用普通意义的点字符 “.”，必须使用转义字符 “\”。

实例 090 鉴别非法电话号码

(实例位置：配套资源\SL\08\090)

实例说明

程序经常需要用户录入用户信息或联系方式，其中有一些数组的格式是固定的，程序处理逻辑也是按照这个格式来实现的，但是由于用户输入的是字符串，其灵活性较大，容易输入错误格式的数据。例如，用户联系信息的电话号码就是固定格式的数据，本实例将演示如何利用正则表达式来确定输入的电话号码格式是否匹配，实例的运行效果如图 8.14 所示，在程序中加入该模块可以禁止用户输入错误的电话号码。



图 8.14 验证电话号码

实现过程

(1) 在项目中新建窗体类 CheckPhoneNum，在该窗体中添加 3 个文本框，用于输入姓名、年龄与电话号码，再添加一个“验证”按钮。

(2) 编写“验证”按钮的事件处理方法，该方法获取用户在文本框中输入的电话号码字符串，然后调用 check() 方法进行验证，并将验证结果通过对话框进行输出。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String text = phoneNumField.getText(); // 获取用户输入
    String info = check(text); // 对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info); // 用对话框输出验证结果
}
```

(3) 编写 check() 方法，该方法用于验证指定的字符串与正确的电话号码格式是否匹配，该方法首先判断字符串是否为空，然后再通过正则表达式对字符串进行验证，并将验证结果作



为方法的返回值。关键代码如下：

```
public String check(String text){  
    if(text == null || text.isEmpty()) {  
        return "请输入电话号码！";  
    }  
    String regex = "^\\d{3}-?\\d{8}||\\d{4}-?\\d{8}$"; // 定义正则表达式  
    if(text.matches(regex)){ // 判断输入数据是否为电话号码  
        return text + "\n 是一个合法的电话号码！";  
    }else{  
        return text + "\n 不是一个合法的电话号码！";  
    }  
}
```



Note

技术要点

本实例使用正则表达式对电话号码进行了格式匹配验证。正则表达式通常被用于判断语句中，来检查某一字符串是否满足某一格式。它是含有一些特殊意义字符的字符串，这些特殊字符称为正则表达式的元字符。例如，“\d”表示字母 0~9 中任何一个。“\d”就是元字符。正则表达式中的元字符及其意义如表 8.1 所示。

表 8.1 正则表达式中的元字符

元字符	正则表达式中的写法	意义
.	"."	代表任意一个字符
\d	"\d"	代表 0~9 的任何一个数字
\D	"\D"	代表任何一个非数字字符
\s	"\s"	代表空白字符，如 '\t'、'\n'
\S	"\S"	代表非空白字符
\w	"\w"	代表可用作标识符的字符，但不包括 "\$" 符
\W	"\W"	代表不可用于标识符的字符
\p{Lower}	代表小写字母{a~z}	
\p{Upper}	\p{Upper}	代表大写字母{A~Z}
\p{ASCII}	\p{ASCII}	ASCII 字符
\p{Alpha}	\p{Alpha}	字母字符
\p{Digit}	\p{Digit}	十进制数字，即[0~9]
\p{Alnum}	\p{Alnum}	数字或字母字符
\p{Punct}	\p{Punct}	标点符号：!"#\$%&'^*+,./;<=>?@[{}]-_`~
\p{Graph}	\p{Graph}	可见字符：[\p{Alnum}\p{Punct}]
\p{Print}	\p{Print}	可打印字符：[\p{Graph}\x20]
\p{Blank}	\p{Blank}	空格或制表符：[\t]
\p{Cntrl}	\p{Cntrl}	控制字符：[\x00-\x1F\x7F]

多学两招：

一个 Java 对象（字符串也是 Java 对象）必须先初始化才能使用，否则编译器会报告“使用的变量未初始化”错误。



实例 091 将字符串转换成整数

(实例位置: 配套资源\SL\08\091)



Note

实例说明

在 Swing 程序中, 用户输入的信息通常是使用 `getText()`方法获得的。该方法的返回值是 `String` 类型。如果用户输入的是一串数字, 而程序又需要使用这些数字进行运算, 则可以将字符串转换成整型或浮点型。本实例将用户的输入转换为整数并计算其平方数。实例的运行效果如图 8.15 所示。

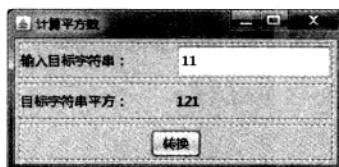


图 8.15 将字符串转换成整数并做运算实现过程

(1) 编写类 `IntegerConversion`, 该类继承了 `JFrame`。在框架中包含了一个文本域用来获得用户的输入、“转换”按钮用于转换用户的输入为整数并计算其平方、显示结果和提示信息的标签。

(2) 编写 `do_buttonActionPerformed()`方法, 用来监听单击“转换”按钮事件。在该方法中, 将用户的输入转换成整数并计算其平方数, 然后将结果显示在标签中。关键代码如下:

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String input = textField.getText(); //获得用户的输入文本
    int number = Integer.parseInt(input); //将文本转换成整数
    label3.setText(number * number + ""); //计算平方数并显示结果
}
```

脚下留神:

此处并未对用户的输入进行校验, 如果用户输入的数不能转换成整数则会出现异常。

指点迷津:

该方法是使用 10 为基数来解析字符串的, 因此结果和字符串的内容相同。

技术要点

`Integer` 类是基本类型中 `int` 类型的包装类, 它可以将基本类型转换成引用类型。在 Java 5.0 版增加了自动装箱和拆箱机制后, 该类的这种用法已经不常用。该类还提供了将字符串转换成 `int` 类型的静态方法。其声明语法如下:

```
public static int parseInt(String s);
```

参数说明

s: 要转换的字符串, 如果不能成功转换则会抛出 `NumberFormatException` 异常。

返回值: 用十进制参数表示的整数值。



多学两招：

除了本实例使用的 `parseInt()` 方法外，API 中还提供了其重载方法。该方法可以根据指定的基数来转换字符串。

语法如下：

```
parseInt(String s,int radix);
```

参数说明

- s：要转换的字符串，如果不能成功转换则会抛出 `NumberFormatException` 异常。
- radix：解析 s 时使用的基数。



Note

实例 092 整数进制转换器

(实例位置：配套资源\SL\08\092)

实例说明

由于计算机的特殊结构，其内部使用二进制数据。为了节约空间，又定义了八进制和十六进制格式来表示二进制数据。一个八进制数可以表示 3 位二进制数，一个十六进制数可以表示 4 位二进制数。而对于普通人而言，使用十进制更加容易阅读。本实例将实现一个简单的进制转换器。实例的运行效果如图 8.16 所示。

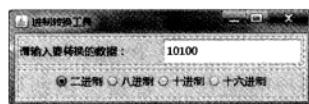


图 8.16 进制转换

脚下留神：

需要先在文本域中输入数字才能进行转换，本实例并未校验空输入的情况。

实现过程

(1) 编写类 `RadixConversion`，该类继承了 `JFrame`。在框架中包含了一个文本域用来获得用户的输入和显示转换的结果，一个按钮组用来实现在不同进制之间的转换功能。

(2) 编写 `do_textField_focusLost()` 方法，用来监听文本域失去焦点事件。在该方法中，使用一个名为 `number` 的域保存用户输入的字符串。关键代码如下：

```
protected void do_textField_focusLost(FocusEvent e) {  
    number = textField.getText();  
} //获得用户的输出
```

指点迷津：

读者可以使用 `String` 类的 `isEmpty()` 方法来判断用户输入是否为空。

(3) 编写 `do_octalRadioButtonActionPerformed()` 方法，用来监听选中“二进制”单选按钮事件。在该方法中，将用户输入的字符串转换成二进制格式并在文本域中显示。关键代码如下：

```
protected void do_binaryRadioButtonActionPerformed(ActionEvent e) {  
    textField.setText(Integer.toBinaryString(Integer.parseInt(number))); //显示转换的结果  
}
```

**指点迷津：**

字符串就是由多个字符组成的，灵活地运用字符数组可以实现复杂的字符串操作，通过数组下标的各种算法可以使字符串更加灵活。

**技术要点**

`Integer` 类设计的初衷是为了在基本类型 `int` 和引用类型之间建立一座桥梁。然而，类库的设计者发现，可以将很多有用的方法也放在该类中。本实例使用其定义的进制转换方法来实现进制转换，使用到的方法如表 8.2 所示。

表 8.2 `Integer` 类的常用方法

方法名	作用
<code>toBinaryString(int i)</code>	返回指定数字 i 的二进制表示形式
<code>toOctalString(int i)</code>	返回指定数字 i 的八进制表示形式
<code>toHexString(int i)</code>	返回指定数字 i 的十六进制表示形式

脚下留神：

以上方法的返回值都是无符号形式的结果，例如 -1 的十六进制表示是 `ffffffff`。

多学两招：

本实例虽然实现了进制转换功能，但是有个缺点：不能进行连续转换。因为 `Integer.parseInt(number)` 代码使用 10 为基数来解析字符串，如果用户先将字符串转换成包含字母的十六进制格式，再转换为其他进制就会出现数字格式异常。解决方法是保存当前数字的基数，请读者自行完成。

实例 093 获取字符串中汉字的个数

(实例位置：配套资源\SL\08\093)

实例说明

字符串中可以包括数字、字母、汉字或者其他字符。使用 `Character` 类的 `isDigit()` 方法可以判断字符串中的某个字符是否为数字，使用 `Character` 类的 `isLetter()` 方法可以判断字符串中的某个字符是否为字母。实例中将介绍一种方法用来判断字符串中的某个字符是否为汉字，通过此方法可以计算字符串中汉字的数量。实例的运行效果如图 8.17 所示。

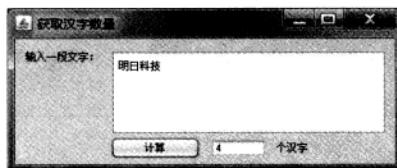


图 8.17 获取字符串中汉字的个数

实现过程

(1) 在项目中创建窗体类 `ChineseAmount`，在窗体中添加接收用户输入的文本域控件、显示汉字数量的文本框控件和计算汉字数量的“计算”按钮。



(2) 编写“计算”按钮的事件处理方法，在该方法中获取用户输入的字符串，然后遍历字符串中的每一个字符，使用正则表达式判断字符是否属于汉字，然后根据判断结果对汉字进行计数，最后把计数结果显示到界面文本框中。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String text = chineseArea.getText(); // 获取用户输入
    int amount = 0; // 创建汉字数量计数器
    for (int i = 0; i < text.length(); i++) { // 遍历字符串每一个字符
        boolean matches = Pattern.matches("^[\\u4E00-\\u9FA5]{0,}$", text.charAt(i) + ""); // 判断字符是否属于汉字编码
        if (matches) { // 如果是汉字
            amount++; // 累加计数器
        }
    }
    numField.setText(amount + ""); // 在文本框中显示汉字数量
}
```

指点迷津：

字符串对象的索引是只读的，只可以读取字符串对象中的字符，不可以根据索引更改字符串中的字符。

技术要点

本实例的关键点在于正则表达式的使用。Java 中提供了 Pattern 用于正则表达式的编译表示形式，该类提供的静态方法 matches() 可以执行正则表达式的匹配。该方法编译给定正则表达式并尝试将给定输入与其匹配。如果要匹配的字符序列与正则表达式匹配则返回 true，否则返回 false。其声明语法如下：

```
public static boolean matches(String regex, CharSequence input);
```

参数说明

- regex：要编译的表达式。
- input：要匹配的字符序列。

多学两招：

使用正则表达式可以非常方便地操作字符串，经常用来验证用户输入的信息，如可以判断用户输入的手机号码格式是否正确，验证用户输入的身份证号码格式是否正确。在本实例中将使用正则表达式来判断字符串中的字符是否为汉字，如果是汉字则计数器加 1，最后得到字符串中所有汉字的数量。

实例 094 批量替换某一类字符串

(实例位置：配套资源\SL\08\094)

实例说明

在字符串操作中，可以使用字符串对象的 split() 方法拆分字符串，还可以使用字符串对象的



`substring()`方法截取一部分字符串。字符串对象为开发者提供了很多方便实用的方法，本实例中将会介绍使用字符串对象的 `replace()`方法替换某一类字符串。实例的运行效果如图 8.18 所示。

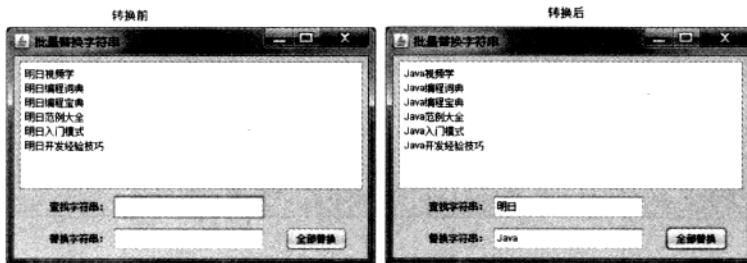


图 8.18 替换字符串前后对比

技术要点

- (1) 在项目中创建窗体类 `StringReplace`。在窗体上创建一个文本框控件，用来显示所有文字；添加两个文本框控件，分别用来接收查找和替换的字符串；再添加一个“全部替换”按钮。
- (2) 编写“全部替换”按钮的事件处理方法，在该方法中获取用户输入的搜索字符串、替换字符串和文本域中的文本字符串，然后执行替换操作，最后将替换结果显示在文本域控件中。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String searchStr = searchTextField.getText();           // 获取搜索字符串
    String replaceStr = replaceTextField.getText();          // 获取替换字符串
    String text = txtArea.getText();                         // 获取段落文本
    String newText = text.replace(searchStr, replaceStr);   // 执行替换
    txtArea.setText(newText);                               // 替换结果显示在文本域控件中
}
```

指点迷津：

字符串就是由多个字符组成的，灵活地运用字符数组可以实现复杂的字符串操作，通过数组下标的各种算法可以使字符串更加灵活。

技术要点

本实例重点在于向读者介绍字符串对象的 `replace()`方法的使用。

使用字符串对象的 `replace()`方法可以方便地替换字符串中指定的内容。由于字符串是不可变的，`replace()`方法会返回一个新的字符串对象。`replace()`方法的使用如图 8.19 所示。



图 8.19 字符串对象 `replace()`方法的使用

从图 8.19 中可以看到，字符串对象调用了 `replace()`方法，此方法将会返回一个被替换内容的新字符串。新字符串的值为“abCDEFg”。





脚下留神：

由于字符串是不可变的，字符串对象调用了 replace()方法后返回的是一个新的字符串而不是原来的字符串。

多学两招：

用字符串对象的 replace()方法可以方便地替换字符串中指定的内容。有一点要注意，replace()方法并不是只替换掉一个匹配的字符串，而是一次性替换掉所有匹配的字符串，由于字符串是不可变的，replace()方法会返回一个新的字符串对象。

实例 095 查看数字的取值范围

(实例位置：配套资源\SL\08\095)

实例说明

Java 是一种强类型语言，每当定义变量时，都需要先指明其类型。对于数字基本类型而言，其取值范围是有限制的。例如，byte 类型的范围是从 -128~127，如果将 128 赋值给一个 byte 类型的数据就会报错。本实例将用来显示各种数字基本类型的最大值和最小值。实例的运行效果如图 8.20 所示。



图 8.20 查看取值范围

实现过程

(1) 编写类 NumberLimitation，该类继承了 JFrame。在框架中包含 4 个标签和一个单选按钮组。两个标签被用来显示最大值和最小值，单选按钮组用来供用户选择要显示的基本类型。

(2) 编写方法 do_byteRadioButtonActionPerformed()，用来监听选中“byte 类型”单选按钮事件。在该方法中，更新了 4 个标签的文本信息，关键代码如下：

```
protected void do_byteRadioButtonActionPerformed(ActionEvent e) {
    maxLabel.setText("byte 类型的最大值: ");
    minLabel.setText("byte 类型的最小值: ");
    maxResult.setText(Byte.MAX_VALUE + "");
    minResult.setText(Byte.MIN_VALUE + "");
}
```

指点迷津：

其他单选按钮的代码与此类似，在此就不做讲解。

技术要点

为了方便基本类型和引用类型之间的转换，Java 为每种基本类型都提供了对应的包装类。现说明如下：byte 的包装类是 Byte、short 的包装类是 Short、int 的包装类是 Integer、long 的包装类是 Long、float 的包装类是 Float、double 的包装类是 Double、boolean 的包装类是 Boolean、char 的包装类是 Character。在各个包装类中，定义了一些常用的域和方法。对于数字基本类型



的包装类而言，其 MAX_VALUE 域表示该类型所能取得的最大值、MIN_VALUE 域表示该类型所能取得的最小值。以 Byte 为例，下面的代码可以获得 byte 类型的最大值：

```
public static final byte MAX_VALUE;
```

对于其他类型而言，代码是类似的。

多学两招：

在 8 种基本类型中，最常用的是 boolean、int 和 double。byte 类型用于输入流和输出流的操作。short、float 通常用于需要节约空间的情况。如果不是与字符编码有关的操作，最好不要使用 char 类型。使用 long 和 float 类型时，需要在数字后面增加 L (l) 和 F (f) 标识。



Note

实例 096 ASCII 编码查看器

(实例位置：配套资源\SL\08\096)

实例说明

ASCII 是 American Standard Code Information Interchange 的缩写。它是基于拉丁字母的一套电脑编码系统，主要用于显示英语字符，是目前世界上最通用的单字节编码。基本的 ASCII 编码包括了 128 个字符。本实例将编写一个 ASCII 编码查看器，可以将字符转换成数字，也可以反向转换。实例的运行效果如图 8.21 所示。



图 8.21 ASCII 编码查看器

实现过程

(1) 编写类 ASCIIViewer，该类继承了 JFrame。在框架中主要包含了两个文本域和两个“转换”按钮，文本域用来获得用户的输入，“转换”按钮用来完成转换功能，并在标签上显示。

(2) 编写 do_toNumberButtonActionPerformed()方法，用来监听单击第一个“转换”按钮事件。在该方法中，将用户输入的字符转换成数字，关键代码如下：

```
protected void do_toNumberButtonActionPerformed(ActionEvent e) {
    String ascii = asciiTextField.getText(); //获得用户输入的字符串
    int i = Character.codePointAt(ascii, 0); //求字符串的第一个字符的代码点
    label3.setText("" + i); //更新标签
}
```

(3) 编写 do_toASCIIButtonActionPerformed()方法，用来监听单击第二个“转换”按钮事件。在该方法中，将用户输入的数字转换成字符，关键代码如下：

```
protected void do_toASCIIButtonActionPerformed(ActionEvent e) {
    String number = numberTextField.getText(); //获得用户输入的字符串
    char[] a = Character.toChars(Integer.parseInt(number)); //求数字所对应的字符数组
    label6.setText(new String(a)); //更新标签
}
```

指点迷津：

如果输入了多个字符，则在转换时只取第一个字符转换成数字。本实例未对数字范围进行校验。



技术要点

Character 类是 char 类型的包装类，该类除了能将 char 类型转换成引用类型外，还包括了大量处理字符编码的方法。本实例使用 codePointAt()方法获得字符的代码点。其声明语法如下：

```
public static int codePointAt(char[] a,int index);
```

参数说明

a: char 数组。

index: 要转换的 char 数组中的 char 值（Unicode 代码单元）的索引。

返回值：给定索引上的 Unicode 代码点。



Note

多学两招：

Character 类的方法和数据是通过 UnicodeData 文件中的信息定义的，该文件是 Unicode Consortium 维护的 Unicode Character Database 的一部分。此文件指定了各种属性，其中包括每个已定义 Unicode 代码点或字符范围的名称和常规类别。此文件及其描述可从 Unicode Consortium 获得。

实例 097 判断手机号的合法性

(实例位置：配套资源\SL\08\097)

实例说明

程序开发中经常需要用户录入某些标准格式的数据，如 E-mail、身份证件、手机号码等。由于这些数据有固定的标准格式，所以程序开发时经常习惯性地不加判断或进行简单判断方便处理数据，但是对于有针对性的程序需要对标准格式中的数据进行详细解析，确认无误后才进行处理。本实例以手机号码为例，使用正则表达式进行匹配判断。实例的运行效果如图 8.22 所示。



图 8.22 判断是否是合法的手机号

实现过程

(1) 在项目中创建窗体类 CheckPhoneNum。在窗体中添加一个接收用户输入的文本框和一个“提交”按钮。

(2) 编写“提交”按钮的事件处理方法，在该方法中获取用户输入的手机号码，然后定义正则表达式，如果用户输入的手机号码能够与这个正则表达式匹配则手机号码格式正确，否则认为是错误的手机号码。关键代码如下：

```
protected void do_buttonActionPerformed(ActionEvent e) {
    String text = textField.getText(); // 获取用户输入号码
    String regex = "^13\\d{9}|15\\d{9}|18\\d{9}$"; // 定义正则表达式
    if (text.matches(regex)) { // 测试匹配结果
        showMessageDialog(null, text + " 是合法的手机号"); // 提示合法手机号码
    } else {
```



```
showMessageDialog(null, text + " 不是合法的手机号码"); //提示非法手机号码
}
}
```

技术要点

本实例的关键点在于手机号的长度与手机前两位数字范围的验证，用户在输入手机号时，程序可以获取的只有字符串类型，所以本实例利用字符串的灵活性与正则表达式搭配进行验证。该方法是 String 字符串类的方法，用于判断字符串与指定的正则表达式是否匹配。其声明语法如下：

```
public boolean matches(String regex);
```

参数说明

regex: 用来匹配此字符串的正则表达式。

返回值：当且仅当此字符串符合给定的正则表达式条件时，返回 true。



Note

多学两招：

由于正则表达式的存在，验证与匹配各种规律的字符串数据更加方便、快捷，所以尽量使用正则表达式控制与限制用户操作初期输入正确的数据，这样可以提高程序数据的价值，使程序更容易控制数据。

实例 098 用字符串构建器追加字符

(实例位置：配套资源\SL\08\098)

实例说明

字符串是程序开发中使用最频繁的数据，在 Java 中字符串是 String 类的对象，它是不可变数据，当执行字符串连接操作时将生成新的字符串，而不是修改原有字符串，所以大量字符串操作非常耗时。本实例分别演示使用 String 类与 StringBuilder 类进行 3 万个字符串追加的操作，并输出其运行时间，结果如图 8.23 所示。从本实例的运行结果中可以看出，普通的字符串连接操作将耗时 2100 多毫秒，而使用 StringBuilder 字符串构建器却在 0.3 毫秒左右的时间完成了 3 万个字符的追加操作。所以对于大量字符串操作，应该使用 StringBuilder 字符串构建器来完成。

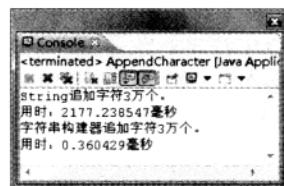


图 8.23 显示追加时间

实现过程

在项目中新建 AppendCharacter 类。在该类的主方法中创建字符串对象 appendStr，并通过循环为该字符串连接 3 万个字符，计算并输出其用时。再创建 StringBuilder 字符串构建器，同样为其追加 3 万个字符，计算并输出用时。关键代码如下：

```
public static void main(String[] args) {
    String appendStr = "";
    long startTime = System.nanoTime();
    for(int i=20000;i<50000;i++){
        //创建字符串变量
        //开始计时
        //遍历 3 万个字符
    }
}
```



```

appendStr+=(char)i; //字符串与每个字符执行连接操作
}
long endTime = System.nanoTime(); //结束计时
System.out.println("String 追加字符 3 万个。");

System.out.println("用时: "+(endTime-startTime)/1000000d+"毫秒"); //输出用时
///////////////////////////////
StringBuilder strBuilder=new StringBuilder(); //创建字符串构建器
startTime = System.nanoTime(); //开始计时
for(int i=20000;i<50000;i++){ //遍历 3 万个字符
    strBuilder.append((char)i); //把每个字符追加到构建器
}
endTime = System.nanoTime(); //结束计时
System.out.println("字符串构建器追加字符 3 万个。");
System.out.print("用时: "+(endTime-startTime)/1000000d+"毫秒"); //输出用时
}

```

多学两招：

StringBuilder 对于线程来说，是不安全的，它适用于单任务的字符串操作，如果把它应用于多线程中将会涉及异步访问的安全性。Java 早期版本提供的 StringBuffer 类可以作为多线程应用的考虑，它是线程安全的，也正因为考虑到线程安全问题，它会比 StringBuilder 稍微慢一些，但是差距很小，读者在开发程序时应灵活运用。

实例 099 去掉字符串中的所有空格

(实例位置：配套资源\SL\08\099)

实例说明

在字符串操作中，可以使用字符串对象的 trim() 方法去除字符串对象前端和后端的所有空格，但是，如果空格在字符串的中间位置出现，使用 trim() 方法是没有效果的，那么怎样才可以有效地去除空格呢？本实例将通过字符串操作实现这个功能。实例的运行效果如图 8.24 所示。

实现过程

- (1) 在项目中创建窗体类 DeleteBlank。在窗体中添加两个文本框和一个“去除空格”按钮。
- (2) 编写“去除空格”按钮的事件处理方法，在该方法中获取用户输入的带有空格的字符串，创建一个字符串构建器用于提取非空格的字符，遍历字符串的每个字符，过滤所有空格，并将非空格字符追加到字符串构建器中，最后将构建器中的字符串显示到文本框中。关键代码如下：

```

protected void do_buttonActionPerformed(ActionEvent e) {
    String text = textField.getText(); //获取用户输入文本
    StringBuilder strBuilder=new StringBuilder(); //创建字符串构建器

```

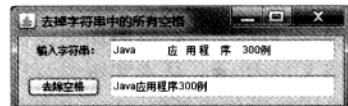


图 8.24 去掉字符串中的所有空格



```

for(int i=0;i<text.length();i++){
    char charAt = text.charAt(i);
    if(charAt==' ')
        continue;
    strBuilder.append(charAt);
}
resultField.setText(strBuilder.toString());
}

```

//遍历字符串
//获取每个字符
//过滤空格字符
//追加非空格字符到字符构建器
//把构建器中的字符串显示到文本框

**Note**

技术要点

本实例重点在于向读者介绍怎样使用 `StringBuilder` 便捷、高效地操作字符串，下面介绍本实例对 `StringBuilder` 构建器的应用。构建器的 `append()` 方法可以向其尾部追加新的字符串。其声明语法如下：

```
append(String str);
```

参数说明

`str`：要向构建器尾部追加的字符串。

返回值：此对象的一个引用。

多学两招：

有些读者认为只要把 Java 类放到某个文件夹（目录）下，这个文件夹就是类的包名，这是一种误解。有文件夹结构不等于有了包名，必须在类的首行代码中通过 `package` 语句指定包的名称而不是靠文件结构类指定。

实例 100 Double 类型的比较

(实例位置：配套资源\SL\08\100)

实例说明

对于 `double` 类型（基本类型）的数据，可以直接使用普通的运算符来进行比较，如“`==`”。然而，对于 `Double` 类型（引用类型）却不行。引用类型如果使用“`==`”来进行比较的话是判断内存地址是否相同，答案通常是否定的。本实例演示如何使用 `Double` 类中定义的方法来进行对象间比较，实例的运行效果如图 8.25 所示。

实现过程

编写类 `DoubleTest`，在该类的 `main()` 方法中定义了两个 `Double` 类型的数据，并使用 `compareTo()` 方法对其进行比较。关键代码如下：

```

public class DoubleTest {
    public static void main(String[] args) {
        Double number1 = 12.4;           //定义 Double 类型的数据
        Double number2 = 12.3;           //定义 Double 类型的数据
        System.out.println("number1: " + number1); //输出定义的数据
    }
}

```

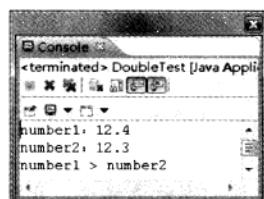


图 8.25 Double 类型的比较



```
System.out.println("number2: " + number2);      //输出定义的数据
switch (number1.compareTo(number2)) {           //判断两个 Double 类型数据之间的关系
    case -1:
        System.out.println("number1 < number2"); //输出比较的结果
        break;
    case 0:
        System.out.println("number1 == number2"); //输出比较的结果
        break;
    case 1:
        System.out.println("number1 > number2"); //输出比较的结果
        break;
}
```



Note

脚下留神：

不要忘记在每个 case 子句的末尾增加 break 来跳出比较。

技术要点

Double 类是基本类型 double 的包装类，该类提供了比较两个 Double 类型对象的 compareTo() 方法。其声明语法如下：

```
public int compareTo(Double anotherDouble);
```

参数说明

anotherDouble：要比较的 Double 值。

返回值：如果 anotherDouble 在数字上等于此 Double，则返回 0；如果此 Double 在数字上小于 anotherDouble，则返回小于 0 的值；如果此 Double 在数字上大于 anotherDouble，则返回大于 0 的值。

指点迷津：

如果仅需要判断是否相等，可以使用 equals()方法，该方法已经被重写了。

多学两招：

所有包装类的对象都不能使用“==”来进行比较，还好每个包装类都定义了 compareTo() 方法，可以使用该方法比较相同类型的对象，然后根据返回值的不同来确定比较的结果。负数代表小于，零代表等于，正数代表大于。另外，包装类重写了从 Object 类继承的 equals()方法，因此可以使用该方法来比较两个引用类型是否相等。

第9章

Java 集合类框架

本章读者可以学到如下实例：

- ▶ 实例 101 用动态数组保存学生姓名
- ▶ 实例 102 用 List 集合传递学生信息
- ▶ 实例 103 Map 集合二级联动
- ▶ 实例 104 不重复随机数组排序
- ▶ 实例 105 for 循环遍历 ArrayList
- ▶ 实例 106 Iterator 遍历 ArrayList
- ▶ 实例 107 ListIterator 逆序遍历 ArrayList
- ▶ 实例 108 制作电子词典
- ▶ 实例 109 制作手机电话簿



实例 101 用动态数组保存学生姓名

(实例位置: 配套资源\SL\09\101)



Note

实例说明

Java 中提供了各种数据集合类,这些类主要用于保存复杂结构的数据,其中 ArrayList 集合可以看作动态数组。它突破了普通数组固定长度的限制,可以随时向数组中添加和移除元素,这将使数组更加灵活。如果要获取普通数组,还可以通过该类的 `toArray()`方法获得。本实例通过 ArrayList 集合类实现了向程序动态添加与删除学生姓名的功能,其中所有数据都保存在 ArrayList 集合的实例对象中。实例的运行效果如图 9.1 所示。

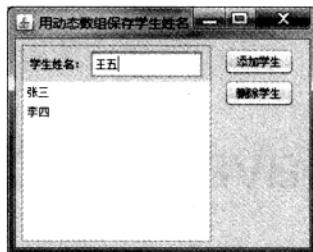


图 9.1 添加删除学生姓名

实现过程

(1) 在项目中新建窗体类 DynamicArray。在窗体中添加文本框控件、列表控件、“添加学生”按钮和“删除学生”按钮。

(2) 编写“添加学生”按钮的事件处理方法,在该方法中获取用户在文本框中输入的字符串,并将这个字符串添加到 ArrayList 集合中,然后调用 `replaceModel()`方法把集合中的数据显示到窗体的列表控件中。关键代码如下:

```
protected void do_buttonActionPerformed(ActionEvent e) {
    textField.requestFocusInWindow();
    textField.selectAll(); //选择文本框文本准备下次输入
    String text = textField.getText(); //获取用户输入姓名
    if (text.isEmpty()) //过滤为输入姓名的情况
        return;
    arraylist.add(text); //把姓名添加到数组集合中
    replaceModel(); //把数组集合中的内容显示到界面列表控件中
}
```

(3) 编写“删除学生”按钮的事件处理方法,在该方法中获取列表控件的当前选择项,然后从 ArrayList 集合中移除这个选择项的值,最后调用 `replaceModel()`方法把集合中的数据显示到窗体的列表控件中。关键代码如下:

```
protected void do_button_1ActionPerformed(ActionEvent e) {
    Object value = list.getSelectedValue(); //获取列表控件的选择项
    arraylist.remove(value); //从数组集合中移除用户的选择项
    replaceModel(); //把数组集合中的内容显示到界面列表控件中
}
```

(4) 编写 `replaceModel()`方法,在该方法中重新设置列表控件的模型,模型要读取 ArrayList 集合的元素并显示到列表控件中。关键代码如下:

```
private void replaceModel() {
    list.setModel(new AbstractListModel() { //为列表控件设置数据模型显示数组集合中的数据
        @Override
        public int getSize() { //获取数组大小
            ...
        }
    });
}
```



```
        return arraylist.size();
    }
    @Override
    public Object getElementAt(int index) {          //获取指定索引元素
        return arraylist.get(index);
    }
});
```



Note

技术要点

本实例使用了 `ArrayList` 集合类的相关操作方法。下面分别介绍程序中对 `ArrayList` 类 API 的引用。

1. 添加元素

`add()`方法可以为数组集合添加任意类型的元素。其声明语法如下：

```
public boolean add(E element);
```

参数说明

element: 要添加到集合中的任意类型的元素值或对象。

返回值：true。

2. 移除元素

`remove()`方法可以移除集合中的指定元素，其中只包含 Object 类型参数的此方法的重载格式可以从集合中移除首次出现指定值的元素。其声明语法如下：

public boolean remove(Object object);

参数说明

object: 要从集合中移除的对象。

返回值：如果此列表包含指定的元素，则返回 true。

多学两招：

`ArrayList` 集合可以看作是一个动态的数组，它比普通数组更加灵活，更适合保存未知数量的数据。例如从数据库中读取指定条件的数据，并且在以后可能会不断地添加新数据，这种情况如果使用普通数组不但会受到长度限制，而且还会受到类型限制，如果采用 `ArrayList` 集合，这些问题就会迎刃而解。

实例 102 用 List 集合传递学生信息

(实例位置: 配套资源\SL\09\102)

实例说明

集合在程序开发中经常用到，如在业务方法中将学生信息、商品信息等存储到集合中然后作为方法的返回值返回给调用者，以此传递大量有序数据。本实例将使用 List 集合在方法之间传递学生信息。实例的运行效果如图 9.2 所示。



图 9.2 将学生信息添加到集中

实现过程

- (1) 在项目中新建窗体类 ClassInfo。在窗体中添加滚动面板，这个面板将放置表格控件。
- (2) 编写 getTable()方法。在该方法中创建表格对象设置表格的数据模型，然后调用 getStudents()方法获取保存学生信息的集合对象，在遍历该集合对象的同时把每个元素添加到表格模型的行，并显示到表格控件中。关键代码如下：

```
private JTable getTable() {
    if (table == null) {
        table = new JTable();
        table.setRowHeight(23);
        String[] columns = {"姓名", "性别", "出生日期"};
        DefaultTableModel model = new DefaultTableModel(columns, 0);
        table.setModel(model);
        List<String> students = getStudents();
        for (String info : students) {
            String[] args = info.split(",");
            model.addRow(args);
        }
    }
    return table;
}
```

- (3) 编写 getStudents()方法，该方法将向调用者传递 List 集合对象，方法中为集合对象添加了多个元素，每个元素值都是一个学生信息，其中包括姓名、性别、出生日期。关键代码如下：

```
private List<String> getStudents() {
    List<String> list = new ArrayList<String>();
    list.add("李哥,男,1981-1-1");
    list.add("小陈,女,1981-1-1");
    list.add("小刘,男,1981-1-1");
    list.add("小张,男,1981-1-1");
    list.add("小董,男,1981-1-1");
    list.add("小吕,男,1981-1-1");
    return list;
}
```

技术要点

实现本实例的关键在于以下几点：

- 将数字格式化，如果存在小数部分，将其转换为 3 位小数到单位厘。
- 分别将整数部分与小数部分转换为大写方式，并插入其单位（亿、万、仟……）。



组合转换后的整数部分与小数部分

多学两招：

List<T>泛型集合表示可通过索引访问对象的强类型列表，它提供用于对列表进行搜索、排序和操作的方法，相对于 ArrayList 类来说，List<T>泛型集合在大多数情况下执行得更好并且是类型安全的。

**Note**

实例 103 Map 集合二级联动

(实例位置：配套资源\SL\09\103)

实例说明

Map 集合可以保存键值映射关系，这非常适合本实例所需要的数据结构，所有省份信息可以保存为 Map 集合的键，而每个键可以保存对应的城市信息，本实例就利用这个 Map 集合实现了省市级联选择框，当选择省份信息时，将改变城市下拉列表框对应的内容。实例的运行效果如图 9.3 所示。

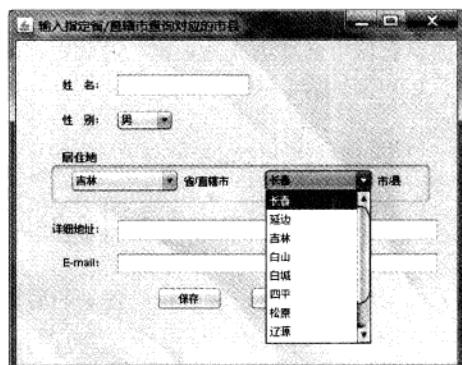


图 9.3 省市级联选择框

实现过程

(1) 在项目中新建窗体类 CityMap。在该类中创建并初始化 Map 集合对象，在该集合对象中保存各省市的关联信息。关键代码如下：

```
public class CityMap {
    public static Map<String, String[]> model=new LinkedHashMap();
    static{
        model.put("北京", new String[]{"北京"});
        model.put("上海", new String[]{"上海"});
        model.put("天津", new String[]{"天津"});
        model.put("重庆", new String[]{"重庆"});
        model.put("黑龙江", new String[]{"哈尔滨","齐齐哈尔","牡丹江","大庆","伊春","双鸭山",
        "鹤岗","鸡西","佳木斯","七台河","黑河","绥化","大兴安岭"});
        model.put("吉林", new String[]{"长春","延边","吉林","白山","白城","四平","松原","辽源",
        "大安","通化"});
    }
}
```



```
model.put("辽宁", new String[]{"沈阳","大连","葫芦岛","旅顺","本溪","抚顺","铁岭","辽  
阳","营口","阜新","朝阳","锦州","丹东","鞍山"});  
//省略类似代码  
}  
}
```

(2) 创建 MainFrame 主窗体类, 在该类中添加 3 个文本框用于输入姓名、详细地址和 E-mail 信息, 添加选择性别的下拉列表框, 添加“保存”和“重置”按钮, 最后添加两个核心控件, 也就是选择省份与选择城市的下拉列表框。

(3) 编写 getProvince()方法, 在该方法中获取 Map 集合的键映射, 也就是省份信息的 Set 集合, 然后将该集合转换为数组, 并作为方法的返回值, 这个方法将在省份下拉列表框的初始化代码时被调用。关键代码如下:

```
public Object[] getProvince() {  
    Map<String, String[]> map = CityMap.model; //获取省份信息保存到 Map 中  
    Set<String> set = map.keySet(); //获取 Map 集合中的键, 并以 Set 集合返回  
    Object[] province = set.toArray(); //转换为数组  
    return province; //返回获取的省份信息  
}
```

(4) 编写选择省份的下拉列表框的事件处理方法, 该方法在省份下拉列表框改变选项时被调用, 方法首先获取下拉列表框的选项值, 然后把该值作为键到 Map 集合中查找对应该键的值, 返回结果是对应省份的所有城市名称组成的数组, 最后用这个数组创建一个数据模型添加到城市下拉列表框控件中, 以更新内容。关键代码如下:

```
private void itemChange() {  
    String selectProvince = (String) comboBox.getSelectedItem();  
    cityComboBox.removeAllItems(); //清空市/县列表  
    String[] arrCity = getCity(selectProvince); //获取市/县  
    cityComboBox.setModel(new DefaultComboBoxModel(arrCity)); //重新添加市/县列表的值  
}
```

(5) 编写 getCity()方法, 该方法主要负责获取对应省份的城市数组, 它将在省份下拉列表框控件的事件处理方法中被调用。关键代码如下:

```
public String[] getCity(String selectProvince) {  
    Map<String, String[]> map = CityMap.model; //获取省份信息保存到 Map 中  
    String[] arrCity = map.get(selectProvince); //获取指定键的值  
    return arrCity; //返回获取的市/县  
}
```

技术要点

本实例的关键技术是 Map 集合的运用。Map 集合可以保存键值对数据, 这样可以根据指定的键名称来获取值数据。

1. 添加映射键值对

本实例通过 Map 集合来保存省市信息, 其中省份作为映射的键, 而城市数组作为键对应的值, Map 映射提供了 put()方法来为集合添加数据。其声明语法如下:

```
put(K key, V value);
```

参数说明

key: 与指定值关联的键。

**Note**

- value: 与指定键关联的值。

2. 获取键对应的值

Map 集合的 get()方法返回指定键所映射的值，如果此映射不包含该键的映射关系，则返回 null 值。其声明语法如下：

```
V get(Object key);
```

参数说明

key: 要返回其关联值的键。

返回值：指定键所映射的值；如果此映射不包含该键的映射关系，则返回 null。

3. 获取键的 Set 集合

Map 集合可以获取所有键的 Set 集合，这个集合中包含 Map 中的所有键，本实例通过 keySet() 方法获取所有键信息来为下拉列表框添加内容。其声明语法如下：

```
Set<K> keySet();
```

多学两招：

Map 集合的具体实现有很多，应该根据需要来选择，其中 HashMap 是最常用的映射集合，它只允许一条记录的键为 null，但是却不限制集合中值为 null 的数量。HashTable 实现一个映射，它不允许任何键值为空。TreeMap 集合将对集合中的键值排序，默认排序方式为升序。

实例 104 不重复随机数组排序

(实例位置：配套资源\SL\09\104)

实例说明

随机数组就是在指定长度的数组中用随机数字为每个元素赋值，这常用于需要不确定数值的环境，如拼图游戏需要随机数组来打乱图片排序。可是随机数的重复问题也同时存在，这个问题也常常被忽略，本实例将利用 TreeSet 集合实现不重复的数列，并自动完成元素的排序然后生成数组。实例的运行效果如图 9.4 所示。

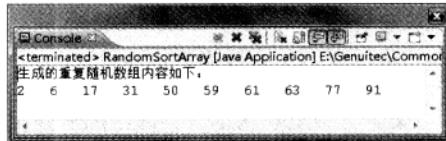


图 9.4 不重复随机数组排序

实现过程

(1) 在项目中新建类 RandomSortArray。

(2) 在类的主方法中创建 TreeSet 集合对象，再创建 Random 随机数对象，然后通过计数器控制循环生成随机数并添加到集合对象中，最后通过集合对象提取数组并显示在控制台中。关键代码如下：

```
public static void main(String[] args) {
    TreeSet<Integer> set = new TreeSet<Integer>(); // 创建 TreeSet 集合对象
    Random ran = new Random(); // 创建随机数对象
    int count = 0; // 定义随机数计数器
    while (count < 10) { // 循环生成随机数
        set.add(ran.nextInt(100)); // 将生成的随机数添加到集合中
    }
    Integer[] arr = set.toArray(new Integer[set.size()]); // 将集合转换为数组
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
}
```



```
boolean succeed = set.add(ran.nextInt(100));
if (succeed) //为集合添加数字
    count++;
}
int size = set.size(); //获取集合大小
Integer[] array = new Integer[size]; //创建同等大小的数组
set.toArray(array); //获取集合中的数组
System.out.println("生成的重复随机数组内容如下: ");
for (int value : array) { //遍历输出数组内容
    System.out.print(value + " ");
}
}
```



Note

技术要点

本实例使用了 TreeSet 集合对象的 API 实现元素的添加以及数组的提取。

1. 添加元素

TreeSet 类的 add()方法可以为集合添加元素，TreeSet 集合属于 Set 集合的子类，Set 集合不允许有重复的元素存在，所以重复数据是不允许添加到 Set 集合中的，而 add()方法的返回值可以确定添加操作是否成功完成。其声明语法如下：

```
public boolean add(E e);
```

参数说明

e：要添加到集合中的任意类型的数据。

返回值：如果此 set 尚未包含指定元素，则返回 true。

2. 提取集合中的数组

Java 的集合对象可以调用 toArray()方法将集合中的所有数据提取到一个新的数组中，本实例就调用了该方法。其声明语法如下：

```
public <T> T[] toArray(T[] array);
```

参数说明

array：保存集合数据的数组。

返回值：如果参数 array 指定的数组长度小于 Set 集合元素的数量，则返回新的可以容纳 Set 集合所有元素的数组；否则返回参数指定的数组对象。

脚下留神：

如果方法指定的数组参数可以容纳下 Set 集合中的所有元素，那么方法的返回值就是这个数组参数，它们用“==”判断结果为 true，因为同一个数组对象的内存地址相等。

如果数组参数长度大于 Set 集合，那么剩余数组元素都赋值为 null，这时要注意可能发生的空指针异常。

多学两招：

本实例首先调用了 size()方法确定 Set 集合的大小，然后创建同等大小的数组，再通过 toArray()方法将集合的所有元素提取到该数组中。更简便的方法是创建一个最小的数组作为方法的参数，这时 toArray()方法会返回与参数数组类型相同的能容纳 Set 集合所有元素的新数组。



实例 105 for 循环遍历 ArrayList

(实例位置: 配套资源\SL\09\105)

实例说明

在使用集合类时,不仅关心容器是如何保存元素的,而且关心如何取出元素。本实例将使用普通 for 循环遍历 ArrayList,从中取出所有序号为奇数的元素。实例的运行效果如图 9.5 所示。

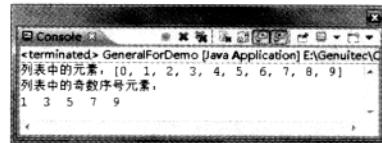


图 9.5 输出序号为奇数的元素

实现过程

- (1) 在 Eclipse 中新建一个项目,在项目中新建类 GeneralForDemo。
- (2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型,并为其添加 10 个元素。

关键代码如下:

```
List<Integer> list = new ArrayList<Integer>(); //创建集合
for (int i = 0; i < 10; i++) { //向列表中增加 10 个元素
    list.add(i);
}
```

- (3) 利用 for 循环遍历 ArrayList 集合,输出列表中序号为奇数的元素。关键代码如下:
- ```
for (int i = 1; i < list.size(); i += 2) { //输出列表中序号为奇数的元素
 System.out.print(list.get(i) + " ");
}
```

### 技术要点

本实例主要应用到了 List 集合中的 get()方法获取列表指定位置的元素。其声明语法如下:

`E get(int index);`

参数说明

index: 要返回元素的索引。

返回值: 列表中指定位置的元素。

### 多学两招:

for 循环中的迭代表达式是用于改变循环条件的语句,根据自己的需要为继续执行循环体语句作准备,如自增、自减等运算。

## 实例 106 Iterator 遍历 ArrayList

(实例位置: 配套资源\SL\09\106)

### 实例说明

ArrayList 在以后的开发中会经常用到,本实例将练习使用 Iterator 和 for 循环组合遍历集合。实例的运行效果如图 9.6 所示。



Note



图 9.6 使用 Iterator 遍历 ArrayList

### 实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建类 IteratorDemo。

(2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素。关键代码如下：

```
List<Integer> list = new ArrayList<Integer>(); //创建集合
for (int i = 0; i < 10; i++) { //向列表中增加 10 个元素
 list.add(i);
}
```

(3) 利用迭代器遍历 ArrayList 集合，其循环条件为如果迭代器中仍有元素可以迭代则继续循环，如果没有则跳出循环。关键代码如下：

```
for(Iterator<Integer> it = list.iterator();it.hasNext();) { //利用迭代器遍历 ArrayList 集合
 System.out.print(it.next()+" ");
}
```

### 技术要点

本实例调用了 Iterator 接口中的 next()方法，返回迭代的下一个元素。其声明语法如下：

**E next();**

返回值：迭代的下一个元素。

## 实例 107 ListIterator 逆序遍历 ArrayList

(实例位置：配套资源\SL\09\107)

### 实例说明

对于列表而言，除了 Iterator，还提供了一个功能更加强大的 ListIterator，它可以实现逆序遍历列表中的元素。本实例将使用其逆序遍历 ArrayList。实例的运行效果如图 9.7 所示。

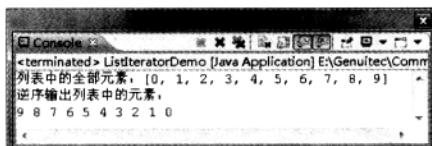


图 9.7 逆序遍历 ArrayList

### 实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建类 ListIteratorDemo。



(2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素。关键代码如下：

```
List<Integer> list = new ArrayList<Integer>(); //创建集合
for (int i = 0; i < 10; i++) { //向列表中增加 10 个元素
 list.add(i);
}
```

(3) 获得迭代器对象，然后利用 hasPrevious()方法逆序输出 ArrayList 集合中的元素。关键代码如下：

```
ListIterator<Integer> li = list.listIterator(); //获得 ListIterator 对象
for (li = list.listIterator(); li.hasNext();) { //将游标定位到列表结尾
 li.next();
}
for (; li.hasPrevious();) { //逆序输出列表中的元素
 System.out.print(li.previous() + " ");
}
```



## Note

### 技术要点

本实例主要应用了 ListIterator 中的 hasPrevious()方法，该方法如果以逆向遍历列表，列表迭代器有多个元素，则返回 true。(换句话说，如果 previous 返回一个元素而不是抛出异常，则返回 true)。其声明语法如下：

```
boolean hasPrevious();
```

返回值：如果以逆向遍历列表，列表迭代器有多个元素，则返回 true。

## 实例 108 制作电子词典

(实例位置：配套资源\SL\09\108)

### 实例说明

词典通常用于解释一个词的含义，这是一种映射关系，因此可以使用 Map 来实现。本实例将制作一个电子词典，实例的运行效果如图 9.8 所示。



图 9.8 电子词典

### 实现过程

(1) 在项目中新建窗体类 DictionaryDemo。在窗体中添加标签等控件。

(2) 编写 do\_this\_windowActivated()方法完成窗体激活事件监听。在该方法中向 Map 中增加数据，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
 words = new HashMap<String, String>();
 words.put("apple", "苹果");
 words.put("banana", "香蕉");
 words.put("water", "水");
}
```

(3) 编写 do\_button\_performed()方法，用来响应按钮单击事件。在该方法中完成了



对单词的查询操作，并根据不同的情况进行提示。关键代码如下：

```

protected void do_buttonActionPerformed(ActionEvent e) {
 String text = textField.getText();
 if (text.isEmpty()) { //获得用户输入的单词
 JOptionPane.showMessageDialog(this, "请输入要查询的单词！", null, JOptionPane.WARNING_MESSAGE);
 return;
 }
 String meaning = words.get(text); //查询单词的含义
 if (meaning == null) { //如果没有这个单词则提示用户
 JOptionPane.showMessageDialog(this, "要查询的单词不存在！", null, JOptionPane.WARNING_MESSAGE);
 return;
 } else { //显示单词的含义
 textArea.setText(meaning);
 }
}

```

## 技术要点

Map 集合中的元素是通过 key、value 进行存储的，要获取集合中指定的 key 或 value 值，需要先通过相应的方法获取 key 或 value 集合，再遍历 key 或 value 集合获取指定值。

## 实例 109 制作手机电话簿

(实例位置：配套资源\SL\09\109)

### 实例说明

为了便于保存联系方式，手机都提供电话簿功能。它也是一种映射关系，因此可以使用 Map 来实现。本实例将制作一个手机电话簿，实例的运行效果如图 9.9 所示。

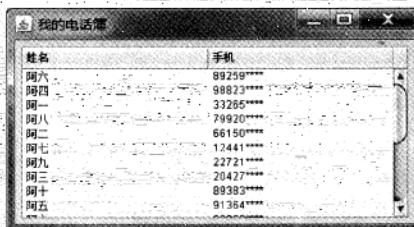


图 9.9 手机电话簿

### 实现过程

- (1) 在项目中新建窗体类 Phonebook。在窗体中添加表格控件。
- (2) 编写 do\_this\_windowActivated()方法完成窗体激活事件监听。在该方法中向表格模型中增加数据，最后更新表格模型。关键代码如下：

```

protected void do_this_windowActivated(WindowEvent e) {
 Map<String, String> directory = new HashMap<String, String>(); //创建集合

```



Note

```

directory.put("阿一", "33265*****");
directory.put("阿二", "66150*****");
directory.put("阿三", "20427*****");
directory.put("阿四", "98823*****");
directory.put("阿五", "91364*****");
directory.put("阿六", "89259*****");
directory.put("阿七", "12441*****");
directory.put("阿八", "79920*****");
directory.put("阿九", "22721*****");
directory.put("阿十", "89383*****");
DefaultTableModel model = (DefaultTableModel) table.getModel();
model.setColumnIdentifiers(new Object[] { "姓名", "手机" });
Set<String> names = directory.keySet();
for (Iterator<String> it = names.iterator(); it.hasNext();) {
 String name = it.next();
 model.addRow(new Object[] { name, directory.get(name) });
}
table.setModel(model);
}

```

//向集合中添加元素  
//获得表格模型  
//设置表头  
//获得键集合  
//获得键  
//向表格中添加元素  
//更新表格模型

## 技术要点

Map 集合中的 put()方法用来向集合中添加指定的 key 与 value 的映射关系，Map 集合中同样提供了集合的常用方法，如 clear()、isEmpty()、size()等，除此之外还包括如表 9.1 所示的常用方法。

表 9.1 Map 集合的常用方法

| 方 法                         | 返 回 值      | 功 能 描 述                         |
|-----------------------------|------------|---------------------------------|
| put(key k , value v)        | Object     | 向集合中添加指定的 key 与 value 的映射关系     |
| containsKey(Object key)     | boolean    | 如果此映射包含指定键的映射关系，则返回 true        |
| containsValue(Object value) | boolean    | 如果此映射将一个或多个键映射到指定值，则返回 true     |
| get(Object key)             | Object     | 如果存在指定的键对象，则返回该对象对应的值，否则返回 null |
| keySet()                    | Set        | 返回该集合中的所有键对象组成的 Set 集合          |
| values()                    | Collection | 返回该集合中所有值对象形成的 Collection 集合    |