

# 字符串和文本I/O

## 学习目标

- 使用String类处理定长的字符串 (9.2节)。
- 使用Character类处理单个字符 (9.3节)。
- 使用StringBuilder/StringBuffer类处理可变长字符串 (9.4节)。
- 区别String、StringBuilder和StringBuffer类 (9.2~9.4节)。
- 学习如何从命令行传递参数给main方法 (9.5节)。
- 使用File类获取文件的属性、删除和重命名文件 (9.6节)。
- 使用PrintWriter类向文件写数据 (9.7.1节)。
- 使用Scanner类从文件读取数据 (9.7.2节)。
- (GUI) 使用对话框打开文件 (9.8节)。

## 9.1 引言

我们经常会遇到涉及字符串处理和文件输入/输出的问题。假如需要编写一个程序，该程序用一个新字替换文件中所有出现某个字的地方。该如何实现这个功能呢？本章介绍字符串和文本文件，它们可以帮助解决此类问题。（因为这里没有引入什么新的概念，所以教师可以将这章布置给学生自学。）

## 9.2 字符串类String

字符串 (string) 是由字符构成的一个序列。在很多语言中，字符串都被当做字符数组来处理，但是在Java中，字符串是一个对象。String类中有11个构造方法以及40多个处理字符串的方法。这不仅在程序设计中非常有用，而且也是一个学习类和对象的很好的例子。

### 9.2.1 构造一个字符串

可以用字符串直接量或字符数组创建一个字符串对象。使用如下语法，从字符串直接量创建一个字符串：

```
String newString = new String(stringLiteral);
```

参数StringLiteral是一个括在双引号内的字符序列。下面的语句为字符串直接量"Welcome to Java"创建一个String对象message：

```
String message = new String("Welcome to Java");
```

Java将字符串直接量看做String对象。所以，下面的语句是合法的：

```
String message = "Welcome to Java";
```

还可以用字符数组创建一个字符串。例如，下述语句构造一个字符串 "Good Day"：

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y'};  
String message = new String(charArray);
```

**注意** String变量存储的是对String对象的引用，String对象里存储的才是字符串的值。严格地讲，术语String变量、String对象和字符串值是不同的。但在大多数情况下，它们之间的区别是可以忽略的。为简单起见，术语字符串通常用来指String变量、String对象和字符串的值。

## 9.2.2 不可变字符串与限定字符串

String对象是不可变的，它的内容是不能改变的。下列代码会改变字符串的内容吗？

```
String s = "Java";
s = "HTML";
```

答案是不能。第一条语句创建了一个内容为“Java”的String对象，并将其引用赋值给s。第二条语句创建了一个内容为“HTML”的新String对象，并将其引用赋值给s。赋值后第一个String对象仍然存在，但是不能再访问它，因为变量s现在指向了新的对象，如图9-1所示。

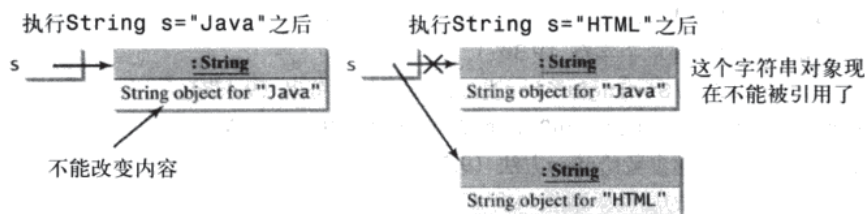
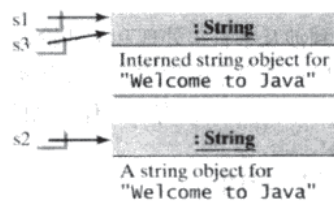


图9-1 字符串是不可变的；一旦创建，它们的内容是不能修改的

因为字符串在程序设计中是不可变的，但同时又会频繁地使用，所以Java虚拟机为了提高效率并节约内存，对具有相同字符串序列的字符串直接量使用同一个实例。这样的实例称为限定的（interned）。例如，下面的语句：

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



程序结果显示：

```
s1 == s2 is false
s1 == s3 is true
```

在上述语句中，由于s1和s3指向相同的限定字符串“Welcome to Java”，因此，s1==s3为true。但是，s1==s2为false，这是因为尽管s1和s2的内容相同，但它们是不同的字符串对象。

## 9.2.3 字符串的比较

String类提供了多种对字符串进行比较的方法，如图9-2所示。

java.lang.String
+equals(s1: String): boolean
+equalsIgnoreCase(s1: String): boolean
+compareTo(s1: String): int
+compareToIgnoreCase(s1: String): int
+regionMatches(index: int, s1: String, s1Index: int, len: int): boolean
+regionMatches(ignoreCase: boolean, index: int, s1: String, s1Index: int, len: int): boolean
+startsWith(prefix: String): boolean
+endsWith(suffix: String): boolean

如果这个字符串等于字符串s1则返回true  
如果不区分大小写这个字符串等于字符串s1则返回true  
返回一个大于0、等于0或小于0的整数以表明这个字符串是大于、等于还是小于s1

除了不区分大小写之外，其他都和compareTo是一样的

如果这个字符串指定的子域精确匹配字符串s1中指定的子域则返回true

除了可以指定匹配是否是区分大小写的，其他都和前一个方法一样

如果这个字符串以指定前缀开始则返回true  
如果这个字符串以指定后缀结束则返回true

图9-2 String类包含多种比较字符串的方法

如何比较两个字符串的内容是否相等呢？你可能会尝试使用==操作符，如下所示：

```
if (string1 == string2)
    System.out.println("string1 and string2 are the same object");
else
    System.out.println("string1 and string2 are different objects");
```

然而，运算符==只能检测string1和string2是否指向同一个对象，但它不会告诉你它们的内容是否相同。因此，不能使用==运算符判断两个字符串变量的内容是否相同。取而代之，应该使用equals方法。例如，可以使用下面给出的代码比较两个字符串：

```
if (string1.equals(string2))
    System.out.println("string1 and string2 have the same contents");
else
    System.out.println("string1 and string2 are not equal");
```

例如，下面的语句先显示true，然后显示false。

```
String s1 = new String("Welcome to Java");
String s2 = "Welcome to Java";
String s3 = "Welcome to C++";
System.out.println(s1.equals(s2)); // true
System.out.println(s1.equals(s3)); // false
```

compareTo方法也用来对两个字符串进行比较。例如，考虑下述代码：

```
s1.compareTo(s2)
```

如果s1与s2相等，那么该方法返回值0；如果按字典序（即以统一码的顺序）s1小于s2，那么方法返回值小于0；如果按字典序s1大于s2，方法返回值大于0。

方法compareTo返回的实际值是依据s1和s2从左到右数第一个不同字符之间的距离得出的。例如，假设s1为"abc"，s2为"abg"，那么s1.compareTo(s2)返回-4。首先比较的是s1与s2中第一个位置的两个字符（a与a）。因为它们相等，所以比较第二个位置的两个字符（b与b）。因为它们也相等，所以比较第三个位置的两个字符（c与g）。由于字符c比字符g小4，所以比较之后返回-4。

**警告** 如果使用像>、>=、<或<=这样的比较运算符比较两个字符串，就会发生语法错误。替代的方法就是使用s1.compareTo(s2)来进行比较。

**注意** 如果两个字符串相等，equals方法返回true；如果它们不等，方法返回false。compareTo方法会根据一个字符串是否等于、大于或小于另一个字符串，分别返回0、正整数或负整数。

String类还提供了对字符串进行比较的方法equalsIgnoreCase、compareToIgnoreCase和regionMatches。当比较两个字符串是否相等时，方法equalsIgnoreCase和compareToIgnoreCase忽略字母的大小写。方法regionMatches比较两个字符串是否部分相等。还可以使用str.startsWith(prefix)来检测字符串str是否以特定前缀prefix开始，使用str.endsWith(suffix)来检测字符串str是否以特定后缀suffix结束。

## 9.2.4 字符串长度、字符以及组合字符串

String类提供获取字符串长度、获取单个字符和连接字符串的方法，如图9-3所示。

java.lang.String	
+length(): int	返回这个字符串的字符个数
+charAt(index: int): char	返回这个字符串的指定下标处的字符
+concat(s1: String): String	返回连接这个字符串和字符串s1所构成的新字符串

图9-3 String类包括了获取字符串长度、获取单个字符和组合字符串的方法

可以调用字符串的length()方法获取它的长度。例如，message.length()返回字符串message的长度。



**警告** `length`是String类的一个方法，但它是数组对象的一个属性。所以，要获取字符串s中的字符个数，必须使用`s.length()`，而要获取数组a中的元素个数，就必须使用`a.length`。

方法`s.charAt(index)`可用于提取字符串s中的某个特定字符，其中下标`index`的取值范围在0到`s.length()-1`之间。例如，`message.charAt(0)`返回字符W，如图9-4所示。

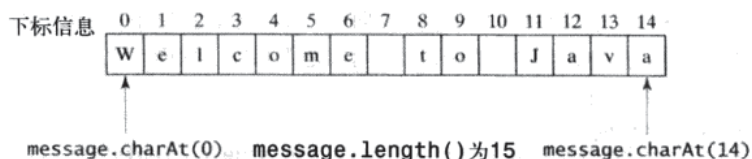


图9-4 String对象在计算机内部是用数组表示的

**注意** 使用一个字符串时，往往是知道它的直接量值的。为方便起见，Java允许在不创建新变量的情况下，使用字符串直接量直接引用字符串。这样，`"Welcome to Java".charAt(0)`是正确的，它返回W。

**注意** 在计算机内部，字符串的值是用私有数组变量表示的。数组是不能从String类的外部访问的。String类提供了许多像`length()`和`charAt(index)`这样的公共方法以获取该数组的信息。这是一个关于封装的很好的例子：类的数据域通过私有修饰符对用户隐藏，这样，用户就不能直接操作数据域。如果数组不是私有的，用户就能通过修改数组而改变字符串的内容，这就会违反String类不可变的原则。

**警告** 在字符串s中越界访问字符是一种常见的程序设计错误。为了避免此类错误，要确保使用的下标不会超过`s.length()-1`。例如，`s.charAt(s.length())`会造成一个`StringIndexOutOfBoundsException`异常。

可以使用`concat`方法连接两个字符串。例如，如下所示的语句将字符串s1和s2连接构成s3：  
`String s3 = s1.concat(s2);`

因为字符串连接在程序设计中应用非常广泛，所以Java提供了一种实现字符串连接的简便办法。可以使用加号(+)连接两个或更多的字符串。因此，上面的语句等价于：

```
String s3 = s1 + s2;
```

下面的代码将字符串`message`、`"and"`和`"HTML"`组合成一个字符串：

```
String myString = message + " and " + "HTML";
```

回顾一下，加号+也可用于连接数字和字符串。在这种情况下，先将数字转换成字符串，然后再进行连接。注意，若要加号实现连接功能，至少要有一个操作数必须为字符串。

## 9.2.5 获取子串

可以使用`charAt`方法从字符串中获取单个字符，如图9-3所示。也可以使用String类中的`substring`方法从字符串中提取子串，如图9-5所示。

<pre>java.lang.String +substring(beginIndex: int): String +substring(beginIndex: int, endIndex: int): String</pre>	<p>返回这个字符串中以指定的<code>beginIndex</code>开始并延续到这个字符串末尾的子串，如图9-6所示</p> <p>返回这个字符串中以指定的<code>beginIndex</code>开始并延续到下标为<code>endIndex-1</code>的字符串的子串，如图9-6所示。注意，在<code>endIndex</code>处的字符不是子串的一部分</p>
--	--

图9-5 String类包含的获取子串的方法

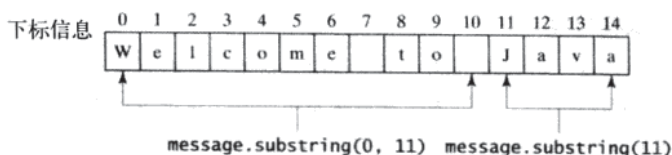


图9-6 substring方法从字符串获取子串

例如：

```
String message = "Welcome to Java".substring(0, 11) + "HTML";
```

现在，字符串message变成"Welcome to HTML"。

**注意** 如果beginIndex就是endIndex，那么substring(beginIndex,endIndex)就会返回一个长度为0的空字符串。如果beginIndex>endIndex，就会出现运行错误。

## 9.2.6 字符串的转换、替换和分隔

String类提供转换、替换和分隔字符串的方法，如图9-7所示。

java.lang.String	
+toLowerCase(): String	返回将所有字符都转换为小写之后的新字符串
+toUpperCase(): String	返回将所有字符都转换为大写之后的新字符串
+trim(): String	返回去掉两端的空白字符之后的新字符串
+replace(oldChar: char, newChar: char): String	返回用一个新字符替换这个字符串中所有和它匹配的字符的新字符串
+replaceFirst(oldString: String, newString: String): String	返回用一个新子串替换这个字符串中第一个和它匹配的子串之后的新字符串
+replaceAll(oldString: String, newString: String): String	返回用一个新子串替换这个字符串中所有和它匹配的子串之后的新字符串
+split(delimiter: String): String[]	返回用定界符分隔的子串所构成的一个字符串数组

图9-7 String类包含转换、替换和分隔字符串的方法

一旦创建了字符串，它的内容就不能改变。但是，方法toLowerCase、toUpperCase、trim、repalce、replaceFirst和replaceAll会返回一个源自原始字符串的新字符串（并未改变原始字符串！）。方法toLowerCase和toUpperCase通过将字符串中所有字符都转换成小写或大写获取一个新字符串。方法trim通过删除字符串两端的空格字符返回一个新字符串。方法replace的版本有好几个，它们实现用新的字符或子串替换字符串中的某个字符或子串。

例如：

"Welcome".toLowerCase() 返回一个新字符串welcome。

"Welcome".toUpperCase() 返回一个新字符串WELCOME。

"Welcome".trim() 返回一个新字符串Welcome。

"Welcome".replace('e', 'A') 返回一个新字符串WAlcoma。

"Welcome".replaceFirst("e", "AB") 返回一个新字符串WABlcome。

"Welcome".replace("e", "AB") 返回一个新字符串WABlcomAB。

"Welcome".replace("el", "AB") 返回一个新字符串WABcome。

split方法可以从一个带特定限定符的字符串中提取标志。例如，下面的代码：

```
String[] tokens = "Java#HTML#Perl".split("#", 0);
for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i] + " ");
```

显示

Java HTML Perl

### 9.2.7 依照模式匹配、替换和分隔

可以通过指定某个模式来匹配、替换或分隔一个字符串。这是一种非常有用且功能强大的特性，通常称为正则表达式 (regular expression)。正则表达式对起步阶段的学生来讲可能会比较复杂。基于这个原因，本节只使用两个简单的模式。若要进行进一步的学习，请参照补充材料III.H。

以String类中的matches方法开始。乍一看，matches方法和equals方法非常相似。例如，下面两条语句的值均为true：

```
"Java".matches("Java");
"Java".equals("Java");
```

但是，matches方法的功能更强大。它不仅可以匹配定长的字符串，还能匹配一套遵从某种模式的字符串。例如，下面语句的结果均为true：

```
"Java is fun".matches("Java.*")
"Java is cool".matches("Java.*")
"Java is powerful".matches("Java.*")
```

在前面语句中的"Java.\*"是一个正则表达式。它描述的字符串模式是以字符串Java开始的，后面紧跟任意0个或多个字符。这里，子串.\*与0个或多个字符相匹配。

方法replaceAll、replaceFirst和split也可以和正则表达式结合在一起使用。例如，下面的语句用字符串NNN替换"a+b\$c"中的\$、+或者#，然后返回一个新字符串。

```
String s = "a+b$c".replaceAll("[${}]", "NNN");
System.out.println(s);
```

这里的正则表达式[`\${}]表示能够匹配\$、+或者#的模式。所以，输出是aNNNbNNNNNNNc。

下面的语句将字符串分隔为由标点符号分隔开的字符串数组。

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]*");
```

```
for (int i = 0; i < tokens.length; i++)
    System.out.println(tokens[i]);
```

这里的正则表达式[.,:;?]\*指定的模式是指匹配.、,、:、;或者?。这里的每个字符都是分隔字符串的分隔符。因此，这个字符串就被分割成Java、C、C#和C++，它们都存储在数组tokens中。

### 9.2.8 找出字符串中的某个字符或者某个子串

String类提供了几个重载的indexOf和lastIndexOf方法，它们可以在字符串中找出一个字符或一个子串，如图9-8所示。

例如，

```
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
```





java.lang.String	
+indexOf(ch: char): int	返回字符串中第一次出现ch的下标。如果不匹配则返回-1
+indexOf(ch: char, fromIndex: int): int	返回字符串中fromIndex之后第一次出现ch的下标。如果不匹配则返回-1
+indexOf(s: String): int	返回字符串中第一次出现字符串s的下标。如果不匹配则返回-1
+indexOf(s: String, fromIndex: int): int	返回字符串中fromIndex之后第一次出现字符串s的下标。如果不匹配则返回-1
+lastIndexOf(ch: int): int	返回字符串中最后一次出现ch的下标。如果不匹配则返回-1
+lastIndexOf(ch: int, fromIndex: int): int	返回字符串中fromIndex之前最后一次出现ch的下标。如果不匹配则返回-1
+lastIndexOf(s: String): int	返回字符串中最后一次出现字符串s的下标。如果不匹配则返回-1
+lastIndexOf(s: String, fromIndex: int): int	返回字符串中fromIndex之前最后一次出现字符串s的下标。如果不匹配则返回-1

图9-8 String类包含匹配子串的方法

### 9.2.9 字符串与数组之间的转换

字符串不是数组，但是字符串可以转换成数组，反之亦然。为了将字符串转换成一个字符数组，可以使用toCharArray方法。例如，下述语句将字符串"Java"转换成一个数组：

```
char[] chars = "Java".toCharArray();
```

因此，chars[0]是'J'，chars[1]是'a'，chars[2]是'v'，chars[3]是'a'。

还可以使用方法getChars(int srcBegin,int srcEnd,char[] dst,int dstBegin)将下标从srcBegin到srcEnd-1的子串复制到字符数组dst中下标从dstBegin开始的位置。例如，下面的代码将字符串"CS3720"中下标从2到6-1的子串"3720"复制到字符数组dst中下标从4开始的位置：

```
char[] dst = {'J', 'A', 'V', 'A', '1', '3', '0', '1'};
"CS3720".getChars(2, 6, dst, 4);
```

这样，dst就变成了{'J', 'A', 'V', 'A', '3', '7', '2', '0'}。

为了将一个字符数组转换成一个字符串，应该使用构造方法String(char[])或者方法valueOf(char[])。例如，下面的语句使用String构造方法由一个数组构造一个字符串：

```
String str = new String(new char[]{'J', 'a', 'v', 'a'});
```

下面的语句使用valueOf方法由一个数组构造一个字符串：

```
String str = String.valueOf(new char[]{'J', 'a', 'v', 'a'});
```

### 9.2.10 将字符和数值转换成字符串

使用静态的valueOf方法能够将字符数组转换成字符串。可以使用valueOf方法的几种重载版本将字符和数值转换成字符串，这些重载的方法的参数类型可以是char、double、long、int和float型，如图9-9所示。

java.lang.String	
+valueOf(c: char): String	返回包含字符c的字符串
+valueOf(data: char[]): String	返回包含数组中字符的字符串
+valueOf(d: double): String	返回表示double值的字符串表述
+valueOf(f: float): String	返回表示float值的字符串表述
+valueOf(i: int): String	返回表示int值的字符串表述
+valueOf(l: long): String	返回表示long值的字符串表述
+valueOf(b: boolean): String	返回表示boolean值的字符串表述

图9-9 String类包含从基本类型值创建字符串的静态方法

例如，为了将double值5.44转换成字符串，可以使用String.valueOf(5.44)。返回值是由字符'5'、'.'、'4'和'4'构成的字符串。

**注意** 可以使用Double.parseDouble(str)或者Integer.parseInt(str)将一个字符串转换成double型值或int型值。

### 9.2.11 格式化字符串

String类包含在String类中的静态format方法，它可以创建一个格式化的字符串。调用该方法的方法是：

```
String.format(format, item1, item2, ..., itemk)
```

这个方法很像printf方法，只是format方法返回一个格式化的字符串，而printf方法显示一个格式化的字符串。例如：

```
String s = String.format("%5.2f", 45.556);
```

创建一个格式化的字符串"45.56"。

### 9.2.12 问题：检测回文串

对于一个字符串，如果从前向后读和从后向前读都是同一个字符串，则称之为回文串。例如，单词“mom”、“dad”和“noon”都是回文串。

这里的问题是编写一个程序，提示用户输入一个字符串，然后报告该字符串是否是回文串。一种解决方案是：先判断该字符串的第一个字符和最后一个字符是否相等。如果相等，检查第二个字符和倒数第二个字符是否相等。这个过程持续进行，直到出现不匹配的情况或者串中所有的字符都检验完毕，当字符串有奇数个字符时，不用检查中间字符。

要实现这个想法，使用两个名为low和high的变量来表示字符串s开始位置和结尾位置的两个字符，如程序清单9-1所示（第22、25行）。初始状态时，low的值为0，high的值为s.length()-1。如果处在这两个位置的字符匹配，就给low加1，同时给high减1（第31~32行）。这个过程一直持续到low>=high或没有出现匹配的情况。

程序清单9-1 CheckPalindrome.java

```
1 import java.util.Scanner;
2
3 public class CheckPalindrome {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        if (isPalindrome(s))
14            System.out.println(s + " is a palindrome");
15        else
16            System.out.println(s + " is not a palindrome");
17    }
18
19    /** Check if a string is a palindrome */
20    public static boolean isPalindrome(String s) {
21        // The index of the first character in the string
22        int low = 0;
23
24        // The index of the last character in the string
25        int high = s.length() - 1;
26    }
```

程序清单9-1

PDG



```

27 while (low < high) {
28     if (s.charAt(low) != s.charAt(high))
29         return false; // Not a palindrome
30
31     low++;
32     high--;
33 }
34
35 return true; // The string is a palindrome
36 }
37 }

```

Enter a string: noon Enter  
noon is a palindrome

Enter a string: moon Enter  
moon is not a palindrome



Scanner类中的nextLine()方法(第11行)读取一行给s.isPalindrome(s),以检查s是否是一个回文串(第13行)。

### 9.2.13 问题: 将十六进制转换为十进制

5.7节给出了一个将一个十进制数转换为十六进制数的程序。本节给出将十六进制数转换为十进制数的程序。

假定一个十六进制数是 $h_n h_{n-1} h_{n-2} \cdots h_2 h_1 h_0$ , 那么等价的十进制数的值为:

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

例如, 十六进制数AB8C是:

$$10 \times 16^3 + 11 \times 16^2 + 8 \times 16^1 + 12 \times 16^0 = 43\ 916$$

程序将提示用户将一个十六进制数作为字符串输入, 然后使用下面的方法将该数转换为一个十进制数:

```
public static int hexToDecimal(String hex)
```

将每个十六进制的字符转换为一个十进制数的穷举方法就是将第i个位置的十六进制数乘以 $16^i$ , 然后将所有这些项相加, 就得到和这个十六进制数等价的十进制数。

注意,

$$\begin{aligned}
 & h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_1 \times 16^1 + h_0 \times 16^0 \\
 & = (\cdots((h_n \times 16 + h_{n-1}) \times 16 + h_{n-2}) \times 16 + \cdots + h_1) \times 16 + h_0
 \end{aligned}$$

这个发现可以导出下面这个将十六进制字符串转换为十进制数的高效算法:

```

int decimalValue = 0;
for (int i = 0; i < hex.length(); i++) {
    char hexChar = hex.charAt(i);
    decimalValue = decimalValue * 16 + hexCharToDecimal(hexChar);
}

```

下面是将算法应用在十六进制数AB8C时各个变量的数值变化情况:

	i	hexChar	hexCharToDecimal(hexChar)	decimalValue
循环开始之前				0
第一次迭代之后	0	A	10	10
第二次迭代之后	1	B	11	$10 \times 16 + 11$
第三次迭代之后	2	8	8	$(10 \times 16 + 11) \times 16 + 8$
第四次迭代之后	3	C	12	$((10 \times 16 + 11) \times 16 + 8) \times 16 + 12$

程序清单9-2给出完整的程序。

程序清单9-2 HexToDecimalConversion.java


```

1 import java.util.Scanner;
2
3 public class HexToDecimalConversion {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a hex number: ");
11        String hex = input.nextLine();
12
13        System.out.println("The decimal value for hex number "
14            + hex + " is " + hexToDecimal(hex.toUpperCase()));
15    }
16
17    public static int hexToDecimal(String hex) {
18        int decimalValue = 0;
19        for (int i = 0; i < hex.length(); i++) {
20            char hexChar = hex.charAt(i);
21            decimalValue = decimalValue * 16 + hexCharToDecimal(hexChar);
22        }
23
24        return decimalValue;
25    }
26
27    public static int hexCharToDecimal(char ch) {
28        if (ch >= 'A' && ch <= 'F')
29            return 10 + ch - 'A';
30        else // ch is '0', '1', ..., or '9'
31            return ch - '0';
32    }
33 }

```

Enter a hex number: AB8C   
The decimal value for hex number AB8C is 43916



Enter a hex number: af71   
The decimal value for hex number af71 is 44913



程序从控制台读取一个字符串（第11行），然后调用hexToDecimal方法，将一个十六进制字符串转换为十进制数（第14行）。字符可以是小写的也可以是大写的。在调用hexToDecimal方法之前将它们都转换成大写的。

在第17~25行定义的hexToDecimal方法返回一个整型值。这个字符串的长度是由第19行调用的hex.length()方法确定的。

在第27~32行定义的方法hexCharToDecimal返回一个十六进制字符的十进制数值。这个字符可以是小写的，也可以是大写的。回忆一下，两个字符的减法就是对它们的统一码做减法。例如，'5'-'0'是5。

### 9.3 字符类Character

Java为每一种基本数据类型都提供了一个包装类。这些类是Character、Boolean、Byte、Short、Integer、Long、Float和Double，它们分别对应基本类型char、boolean、byte、short、int、

long、float和double。所有这些类都在java.lang包中。它们把基本类型数据值当作对象处理。它们还包含一些有用的处理基本数据值的方法。本节介绍Character类。其他包装类将在第14章中介绍。

Character类有一个构造方法和多个确定字符类别（大写字母、小写字母、数字等）的方法，以及将大写字母转换成小写字母或者将小写字母转换成大写字母的方法，如图9-10所示。

可以用一个char值创建一个Character对象。例如，下面的语句为字符'a'创建一个Character对象。

```
Character character = new Character('a');
```

charValue方法返回包装在Character对象中的字符值。方法compareTo将该字符与另一个字符进行比较，返回一个整数值，这个整数值是该字符与另一个字符统一码的差值。当且仅当两个字符相同时，equals方法才返回true。例如，假设charObject为new Character('b')，则：

```
charObject.compareTo(new Character('a'))  返回1
charObject.compareTo(new Character('b'))  返回0
charObject.compareTo(new Character('c'))  返回-1
charObject.compareTo(new Character('d'))  返回-2
charObject.equals(new Character('b'))     返回true
charObject.equals(new Character('d'))     返回false
```

java.lang.Character	
<pre>+Character(value: char) +charValue(): char +compareTo(anotherCharacter: Character): int +equals(anotherCharacter: Character): boolean +isDigit(ch: char): boolean +isLetter(ch: char): boolean +isLetterOrDigit(ch: char): boolean +isLowerCase(ch: char): boolean +isUpperCase(ch: char): boolean +toLowerCase(ch: char): char +toUpperCase(ch: char): char</pre>	<p>使用char值构建一个字符对象 从这个对象返回char值 将这个字符和其他字符进行比较 如果这个字符等于另一个字符则返回true 如果指定字符是一个数字则返回true 如果指定字符是一个字母则返回true 如果指定字符是一个字母或者数字则返回true 如果指定字符是一个小写字母则返回true 如果指定字符是一个大写字母则返回true 返回指定字母的小写 返回指定字母的大写</p>

图9-10 Character类提供处理字符的方法

Character类中的大多数方法都是静态方法。如果字符是一个数字，方法isDigit(char ch)返回true。如果字符是一个字母，方法isLetter(char ch)返回true。如果字符是一个字母或数字，方法isLetterOrDigit(char ch)返回true。如果字符是一个小写字母，方法isLowerCase(char ch)返回true。如果字符是一个大写字母时，方法isUpperCase(char ch)返回true。方法toLowerCase(char ch)返回这个字符的小写字母，方法toUpperCase(char ch)返回这个字符的大写字母。

### 问题：统计字符串中的每个字母

这里的问题是编写一个程序，提示用户输入一个字符串，然后统计在忽略字母大小写的情况下，字符串中每个字母出现的次数。

下面是解决这个问题的步骤：

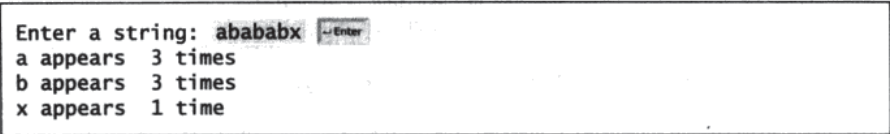
- 1) 使用String类中的toLowerCase方法将字符串中的大写字母转换成小写形式。
  - 2) 创建一个由26个int值构成的数组counts，数组的每个元素记录一个字母出现的次数。也就是说，counts[0]记录a出现的次数，counts[1]记录b出现的次数，依此类推。
  - 3) 对字符串中的每一个字符，判断其是否为小写字母。如果是，则给数组中对应的计数器加1。
- 程序清单9-3给出完整的程序。

程序清单9-3 CountEachLetter.java

```

1 import java.util.Scanner;
2
3 public class CountEachLetter {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        // Invoke the countLetters method to count each letter
14        int[] counts = countLetters(s.toLowerCase());
15
16        // Display results
17        for (int i = 0; i < counts.length; i++) {
18            if (counts[i] != 0)
19                System.out.println((char)('a' + i) + " appears " +
20                    counts[i] + ((counts[i] == 1) ? " time" : " times"));
21        }
22    }
23
24    /** Count each letter in the string */
25    public static int[] countLetters(String s) {
26        int[] counts = new int[26];
27
28        for (int i = 0; i < s.length(); i++) {
29            if (Character.isLetter(s.charAt(i)))
30                counts[s.charAt(i) - 'a']++;
31        }
32
33        return counts;
34    }
35 }

```



```

Enter a string: abababx
a appears 3 times
b appears 3 times
x appears 1 time

```



main方法读取一行（第11行），然后调用countLetters方法（第14行）统计每个字母在该字符串中出现的次数。由于字母的大小写是忽略的，因此，程序使用toLowerCase方法将字符串转换成所有字母都是小写字母，然后将这个新串传递给countLetters方法。

countLetters方法（第25~34行）返回一个26个元素构成的数组，每个元素统计字符串s中一个字母出现的次数。该方法处理字符串中的每个字符。如果字符为字母，其对应的计数器增加1。例如，如果字符（s.charAt(i)）为'a'，则对应的计数器为counts['a'-'a']（也就是counts[0]）。如果字符为'b'，则对应的计数器为counts['b'-'a']（也就是counts[1]），因为'b'的统一码值比'a'的统一码大1。如果字符为'z'，则对应的计数器为counts['z'-'a']（也就是counts[25]），因为'z'的统一码值比'a'的统一码大25。

## 9.4 StringBuilder/StringBuffer类

StringBuilder/StringBuffer类是可以替代String类的另一种处理字符串的解决方案。一般来说，只要使用字符串的地方，都可以使用StringBuilder/StringBuffer类。



`StringBuilder`/`StringBuffer`类比`String`类更灵活。可以给一个`StringBuilder`或`StringBuffer`中添加、插入或追加新的内容，但是`String`对象一旦创建，它的值就确定了。

除了`StringBuffer`中修改缓冲区的方法是同步的之外，`StringBuilder`类与`StringBuffer`类是很相似的。如果是多任务并发访问，就使用`StringBuffer`；而如果是单任务访问，使用`StringBuilder`会更有效。`StringBuffer`和`StringBuilder`中的构造方法和其他方法几乎是完全一样的。本节介绍`StringBuilder`。可以用`StringBuffer`替换`StringBuilder`。程序可以不经任何修改进行编译和运行。

`StringBuilder`类有3个构造方法和30多个用于管理生成器或修改生成器内字符串的方法。可以使用构造方法创建一个空的字符串生成器或从一个字符串创建一个字符串生成器，如图9-11所示。

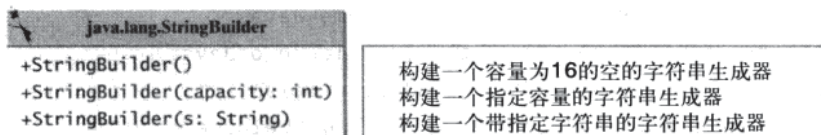


图9-11 `StringBuilder`类包含创建`StringBuilder`实例的构造方法

#### 9.4.1 修改`StringBuilder`中的字符串

可以使用图9-12中列出的方法，在字符串生成器的末尾追加新内容，在字符串生成器的特定位置插入新的内容，还可以删除或替换字符串生成器中的字符：

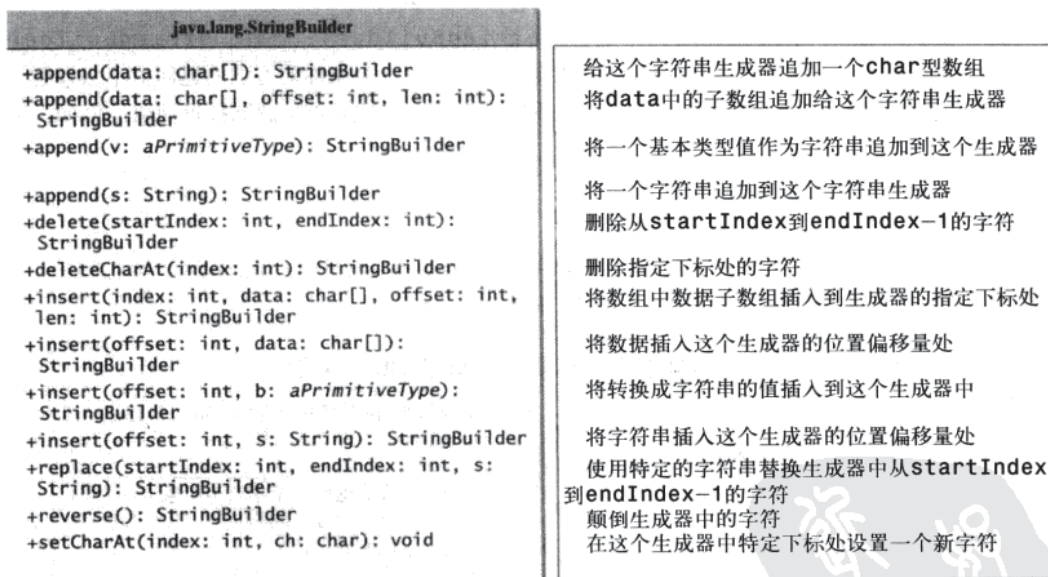


图9-12 `StringBuilder`类包含修改字符串生成器的方法

`StringBuilder`类提供了几个重载方法，可以将`boolean`、`char`、`char`数组、`double`、`float`、`int`、`long`和`String`类型值追加到字符串生成器。例如，下面的代码将字符串和字符追加到`StringBuilder`，构成新的字符串“Welcome to Java”。

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");
```

`StringBuilder`类也包括几个重载的方法,可以将`boolean`、`char`、`char`数组、`double`、`float`、`int`、`long`和`String`类型值插入到字符串生成器。考虑下面的代码:

```
stringBuilder.insert(11, "HTML and ");
```

insert

假设在应用`insert`方法之前, `stringBuilder`包含的字符串是"Welcome to Java"。上面的代码就在`stringBuilder`的第11个位置(就在J之前)插入"HTML and"。新的`stringBuilder`就变成"Welcome to HTML and Java"。

也可以使用两个`delete`方法将字符从生成器中的字符串中删除,使用`reverse`方法倒置字符串,使用`replace`方法替换字符串中的字符,或者使用`setCharAt`方法在字符串中设置一个新字符。

例如,假设在应用下面的每个方法之前, `stringBuilder`包含的是"Welcome to Java"。

`stringBuilder.delete(8,11)`将生成器变为Welcome Java。

`stringBuilder.deleteCharAt(8)`将生成器变为Welcome o Java。

`stringBuilder.reverse()`将生成器变为javaJ ot emocleW。

`stringBuilder.replace(11,15,"HTML")`将生成器变为Welcome to HTML。

`stringBuilder.setCharAt(0,'w')`将生成器变为welcome to Java。

除了`setCharAt`方法之外,所有这些进行修改的方法都做两件事:

- 改变字符串生成器的内容。
- 返回字符串生成器的引用。

例如,下面的语句:

```
StringBuilder stringBuilder1 = stringBuilder.reverse();
```

将生成器中的字符倒置并把生成器的引用赋值给`stringBuilder1`。这样, `stringBuilder`和`stringBuilder1`都指向同一个`StringBuffer`对象。回顾一下,如果你对方法的返回值不感兴趣,所有带返回值类的方法都可以被当作语句调用。在这种情况下,Java就简单地忽略掉返回值。例如,下面的语句

```
stringBuilder.reverse();
```

它的返回值就被忽略了。

**提示** 如果一个字符串不需要任何改变,则使用`String`类而不使用`StringBuffer`类。Java可以完成对`String`类的优化,例如,共享限定字符串等。

#### 9.4.2 `toString`、`capacity`、`length`、`setLength`和`charAt`方法

`StringBuffer`类提供了许多其他处理字符串生成器和获取它的属性的方法,如图9-13所示。

java.lang.StringBuilder	
<pre> +toString(): String +capacity(): int +charAt(index: int): char +length(): int +setLength(newLength: int): void +substring(startIndex: int): String +substring(startIndex: int, endIndex: int): String +trimToSize(): void </pre>	<p>从字符串生成器返回一个字符串对象</p> <p>返回这个字符串生成器的容量</p> <p>返回指定下标处的字符</p> <p>返回这个生成器中的字符个数</p> <p>在这个生成器中设置新的长度</p> <p>返回从<code>startIndex</code>开始的子串</p> <p>返回从<code>startIndex</code>到<code>endIndex-1</code>的子串</p> <p>减少字符串生成器所使用的存储大小</p>

图9-13 `StringBuilder`类包括修改字符串生成器的方法

`capacity()`方法返回字符串生成器当前的容量。容量是指在不增加生成器大小的情况下,能够存储的字符数量。

`length()`方法返回字符串生成器中实际存储的字符数量。`setLength(newLength)`方法设置字符

串生成器的长度。如果参数newLength小于字符串生成器的当前长度，则字符串生成器会被截短到恰好能包含由参数newLength给定的字符个数。如果参数newLength大于或等于当前长度，则给字符串生成器追加足够多的null字符（'\u0000'），使其长度length变成新参数newLength。参数newLength必须大于或等于0。

charAt(index)方法返回字符串生成器中下标为某个特定下标index的字符。下标是基于0的，字符串生成器中的第一个字符的下标为0，第二个字符的下标为1，依此类推。参数index必须大于或等于0，并且小于字符串生成器的长度。

**注意** 字符串的长度总是小于或等于生成器的容量。长度是存储在生成器中的字符串的实际大小，而容量是生成器的当前大小。如果有更多的字符添加到字符串生成器，超出它的容量，则生成器的容量就会自动增加。在计算机内部，字符串生成器是一个字符数组，因此，生成器的容量就是数组的大小。如果超出生成器的容量，就用新的数组替换现有数组。新数组的大小为 $2 \times$ （前一个数组的长度+1）。

**提示** 可以使用new StringBuilder(initialCapacity)创建指定初始容量的StringBuilder。仔细选择初始容量，能够使程序更有效。如果容量总是超过生成器的实际长度，JVM将永远不需要为生成器重新分配内存。另一方面，如果容量过大，将会浪费内存空间。可以使用trimToSize()方法将容量降到实际的大小。

### 9.4.3 问题：忽略既非字母又非数字的字符，判断回文串

程序清单9-1考虑字符串中的所有字符来检测字符串是否是回文串。编写一个新程序，检测一个字符串在忽略掉非字母和非数字的字符时，它是否是一个回文串。

下面是解决这个问题的步骤：

1) 通过删除非字母和非数字字符过滤这个字符串。要做到这一点，需要创建一个空字符串生成器，将字符串中每一个字母或数字字符添加到字符串生成器中，然后从这个生成器返回所求的字符串。可以使用Character类中的isLetterOrDigit(ch)方法来检测字符ch是否是字母或数字。

2) 倒置过滤后的字符串得到一个新字符串。使用equals方法对倒置后的字符串和过滤后的字符串进行比较。

完整的程序如程序清单9-4所示。

程序清单9-4 PalindromeIgnoreNonAlphanumeric.java

```
1 import java.util.Scanner;
2
3 public class PalindromeIgnoreNonAlphanumeric {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        // Display result
14        System.out.println("Ignoring nonalphanumeric characters, \"is "
15            + s + " a palindrome? \" + isPalindrome(s));
16    }
17
18    /** Return true if a string is a palindrome */
19    public static boolean isPalindrome(String s) {
20        // Create a new string by eliminating nonalphanumeric chars
21        String s1 = filter(s);
22    }
```







```

23    // Create a new string that is the reversal of s1
24    String s2 = reverse(s1);
25
26    // Compare if the reversal is the same as the original string
27    return s2.equals(s1);
28 }
29
30 /** Create a new string by eliminating nonalphanumeric chars */
31 public static String filter(String s) {
32     // Create a string builder
33     StringBuilder stringBuilder = new StringBuilder();
34
35     // Examine each char in the string to skip alphanumeric char
36     for (int i = 0; i < s.length(); i++) {
37         if (Character.isLetterOrDigit(s.charAt(i))) {
38             stringBuilder.append(s.charAt(i));
39         }
40     }
41
42     // Return a new filtered string
43     return stringBuilder.toString();
44 }
45
46 /** Create a new string by reversing a specified string */
47 public static String reverse(String s) {
48     StringBuilder stringBuilder = new StringBuilder(s);
49     stringBuilder.reverse(); // Invoke reverse in StringBuilder
50     return stringBuilder.toString();
51 }
52 }

```

Enter a string: ab<>cb?a   
 Ignoring nonalphanumeric characters,  
 is ab<>cb?a a palindrome? true

Enter a string: abcc<>?cab   
 Ignoring nonalphanumeric characters,  
 is abcc<>?cab a palindrome? false



`filter(String s)`方法(第31~44行)逐个地检测字符串`s`中的每个字符,如果字符是字母或数字字符,就将它复制到字符串生成器。该方法返回生成器中的字符串。`reverse(String s)`方法(第47~52行)创建一个新字符串,这个新串是对给定字符串`s`的倒置。`filter`方法和`reverse`方法都会返回一个新字符串。原始字符串并没有改变。

程序清单9-1中的程序通过比较字符串两端的一对字符来检测一个字符串是否是回文串。程序清单9-4使用`StringBuilder`类中的`reverse`方法倒置字符串,然后比较两个字符串是否相等以判断原始字符串是否是回文串。

## 9.5 命令行参数

你或许已经注意到,`main`方法的声明与众不同,它具有`String[]`类型参数`args`。很明显,参数`args`是一个字符串数组。`main`方法就像一个带参数的普通方法。可以通过传递实参来调用一个普通方法。那能给`main`传递参数吗?当然可以。例如,在`TestMain`类中的`main`方法是被`A`中的方法调用的,如下所示:



```
public class A {
    public static void main(String[] args) {
        String[] strings = {"New York",
                            "Boston", "Atlanta"};
        TestMain.main(strings);
    }
}
```

```
public class TestMain {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

main方法就和普通方法一样。此外，还可以从命令行传送参数。

### 9.5.1 向main方法传递字符串

运行程序时，可以从命令行给main方法传递字符串参数。例如，下面的命令行用三个字符串arg0、arg1、arg2启动程序TestMain：

```
java TestMain arg0 arg1 arg2
```

其中，参数arg0、arg1和arg2都是字符串，但是在命令行中出现时，用双引号括住它们是没有必要的。这些字符串用空格分隔。如果字符串包含空格，那就必须使用双引号括住。考虑下面的命令行：

```
java TestMain "First num" alpha 53
```

它用三个字符串"First num"、alpha和53启动这个程序，其中53是一个数值字符串。因为"First num"是一个字符串，所以要用双括号括住它们。注意，53实际上是当作字符串处理的。在命令行可以使用"53"来代替53。

当调用main方法时，Java解释器会创建一个数组存储命令行参数，然后将该数组的引用传递给args。例如，如果调用具有n个参数的程序，Java解释器创建一个如下所示的数组：

```
args = new String[n];
```

然后，Java解释器传递参数args去调用main方法。

**注意** 如果运行程序时没有传递字符串，那么使用new String[0]创建数组。在这种情况下，该数组是长度为0的空数组。args是对这个空数组的引用。因此，args不是null，但是args.length是0。

### 9.5.2 问题：计算器

假设要开发一个程序，完成整型数的算术运算。程序接收三个参数：一个整数、紧随其后的一个运算符以及另一个整数。例如，使用下面的命令对两个整数进行相加：

```
java Calculator 2 + 3
```

程序将显示下面的输出：

```
2 + 3 = 5
```

图9-14显示这个程序的示例运行。

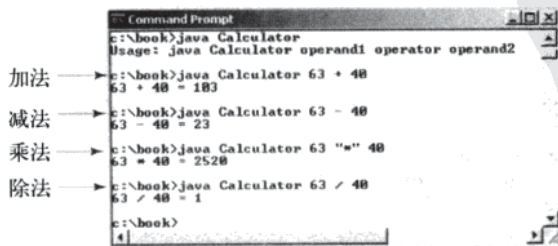


图9-14 程序从命令行获取三个参数（操作数1 运算符 操作数2），然后显示这个算术运算表达式以及算术运算的结果

传递给main程序的字符串存储在字符串数组args中。第一个字符串存储在arg[0]中，

`args.length`是传入的字符串个数。

下面是程序的步骤：

1) 利用`args.length`判断命令行是否提供了三个参数。如果没有，就使用`System.exit(0)`结束程序。

2) 运用`args[1]`中指定的运算符完成对操作数`args[0]`和`args[2]`的二元运算。

这个程序如程序清单9-5所示。

程序清单9-5 **Calculator.java**

```

1 public class Calculator {
2     /** Main method */
3     public static void main(String[] args) {
4         // Check number of strings passed
5         if (args.length != 3) {
6             System.out.println(
7                 "Usage: java Calculator operand1 operator operand2");
8             System.exit(0);
9         }
10
11        // The result of the operation
12        int result = 0;
13
14        // Determine the operator
15        switch (args[1].charAt(0)) {
16            case '+': result = Integer.parseInt(args[0]) +
17                    Integer.parseInt(args[2]);
18                break;
19            case '-': result = Integer.parseInt(args[0]) -
20                    Integer.parseInt(args[2]);
21                break;
22            case '*': result = Integer.parseInt(args[0]) *
23                    Integer.parseInt(args[2]);
24                break;
25            case '/': result = Integer.parseInt(args[0]) /
26                    Integer.parseInt(args[2]);
27        }
28
29        // Display result
30        System.out.println(args[0] + ' ' + args[1] + ' ' + args[2]
31            + " = " + result);
32    }
33 }

```

`Integer.parseInt(args[0])` (第16行) 将一个数字字符串转换为一个整数。该字符串必须由数字构成，否则，程序会非正常中断。

**注意** 在运行示例中，必须在命令行中用`"*"`代替`*`：

```
java Calculator 63 "*" 40
```

符号`*`用于命令行时，表示当前目录下的所有文件。所以，为了明确说明乘法运算符，在命令行中必须用双引号括住符号`*`。在使用命令`java Test *`之后，下面的程序就会显示当前目录下的所有文件：

```

public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}

```

## 9.6 文件类File

存储在变量、数组和对象中的数据是暂时的，当程序终止时它们就会丢失。为了能够永久地保存程

序中创建的数据，需要将它们存储到磁盘或光盘上的文件中。这些文件可以传送，也可以随后被其他程序使用。由于数据存储在文件中，所以本节就介绍如何使用File类获取文件的属性以及删除和重命名文件。9.7节将介绍如何从（向）一个文本文件读（写）数据。

在文件系统中，每个文件都存放在一个目录下。绝对文件名（absolute file name）是由文件名和它的完整路径以及驱动器字母组成。例如，c:\book\Welcome.java是文件Welcome.java在Windows操作系统上的绝对文件名。这里的c:\book称为该文件的目录路径（directory path）。绝对文件名是依赖机器的。在UNIX平台上，绝对文件名可能会是/home/liang/book/Welcome.java，其中/home/liang/book是文件Welcome.java的目录路径。

File类特意提供了一种抽象，这种抽象是指以不依赖机器的方式来处理很多文件和路径名依赖机器的复杂问题。File类包含许多获取文件属性的方法以及重命名和删除文件的方法，如图9-15所示。但是，File类不包含读写文件内容的方法。

java.io.File	
+File(pathname: String)	为特定路径名创建一个File对象。这里的路径名可以是一个目录也可以是一个文件
+File(parent: String, child: String)	为目录parent下的child创建一个File对象。这个child可以是一个文件名也可以是一个子目录
+File(parent: File, child: String)	为目录parent下的child创建一个File对象。这个parent是一个File对象。
+exists(): boolean	在前面的构造方法中，parent是一个字符串
+canRead(): boolean	如果File对象表示的文件或目录存在则返回true
+canWrite(): boolean	如果File对象表示的文件存在且可读则返回true
+isDirectory(): boolean	如果File对象表示的文件存在且可写则返回true
+isFile(): boolean	如果File对象表示一个目录则返回true
+isAbsolute(): boolean	如果File对象表示一个文件则返回true
+isHidden(): boolean	如果使用绝对路径名创建一个File对象则返回true
+getAbsolutePath(): String	如果File对象中表示的文件被隐藏则返回true。被隐藏的确切定义是依赖系统的。在Windows系统中，可以在文件属性对话框中标记该文件为隐藏的。在UNIX系统中，如果文件名以句点字符"."开始则隐藏该文件
+getCanonicalPath(): String	返回File对象表示的完整的路径名或目录名
+getName(): String	除了它能从路径名中删除像"."和".."这样的冗余名，能解析符号链接（在UNIX平台上），能将驱动器号转换为标准的大写字母（在Windows平台上），返回的都是和getAbsolutePath()一样的
+getPath(): String	返回File对象表示的完整的路径名和文件名的最后一个名字。例如：new File("c:\\book\\test.dat").getName()返回test.dat
+getParent(): String	返回File对象表示的完整的路径名和文件名。例如：new File("c:\\book\\test.dat").getPath()返回c:\\book\\test.dat
+lastModified(): long	返回File对象表示的当前路径名和文件名的完整父目录。例如：new File("c:\\book\\test.dat").getParent()返回c:\\book
+length(): long	返回文件最后一次修改的时间
+listFile(): File[]	返回文件的大小，如果文件不存在则返回0，如果是目录则返回目录大小
+delete(): boolean	返回一个File对象目录下的文件
+renameTo(dest: File): boolean	删除这个文件。如果删除成功则该方法返回true
	重命名这个文件。如果操作成功则该方法返回true

图9-15 File类可以用来获取文件和目录属性以及删除和重命名文件

文件名是一个字符串。File类是文件名及其目录路径的一个包装类。例如，在Windows中，语句 new File("c:\\book") 在目录 c:\\book 下创建一个File对象，而语句 new File("c:\\book\\test.dat") 为文件 c:\\book\\test.dat 创建一个File对象。可以用File类的 isDirectory() 方法来判断这个对象是否表示一个目录，还可以用 isFile() 方法来判断这个对象是否表示一个文件名。

**警告** 在Windows中目录的分隔符是反斜杠（\）。但是在Java中，反斜杠是一个特殊的字符，应该写成\\的形式（参见表2-6）。

**注意** 构建一个File实例并不会在机器上创建一个文件。不管文件是否存在，都可以创建任意文件名的File实例。可以调用File实例上的exists()方法来判断这个文件是否存在。



在程序中，不要直接使用绝对文件名。如果使用了像“c:\\book\\Welcome.java”之类的文件名，那么它能在Windows上工作，但是不能在其他平台上工作。应该使用与当前目录相关的文件名。例如，可以使用`new File("Welcome.java")`为在当前目录下的文件Welcome.java创建一个File对象。可以使用`new File("image/us.gif")`为在当前目录下的image目录下的文件us.gif创建一个File对象。斜杠(/)是Java的目录分隔符，这点和UNIX是一样的。语句`new File("image/us.gif")`在Windows、UNIX或任何其他系统上都能工作。

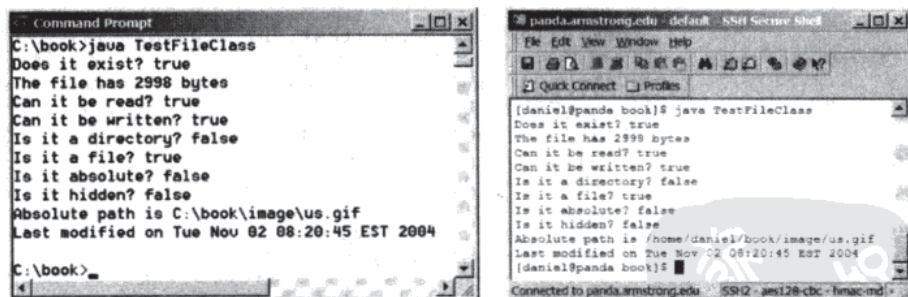
程序清单9-6演示如何创建一个File对象，以及如何使用File类中的方法获取它的属性。这个程序为文件us.gif创建了一个File对象。这个文件存储在当前目录的image目录下。

程序清单9-6 TestFileClass.java

```
1 public class TestFileClass {
2     public static void main(String[] args) {
3         java.io.File file = new java.io.File("image/us.gif");
4         System.out.println("Does it exist? " + file.exists());
5         System.out.println("The file has " + file.length() + " bytes");
6         System.out.println("Can it be read? " + file.canRead());
7         System.out.println("Can it be written? " + file.canWrite());
8         System.out.println("Is it a directory? " + file.isDirectory());
9         System.out.println("Is it a file? " + file.isFile());
10        System.out.println("Is it absolute? " + file.isAbsolute());
11        System.out.println("Is it hidden? " + file.isHidden());
12        System.out.println("Absolute path is " +
13            file.getAbsolutePath());
14        System.out.println("Last modified on " +
15            new java.util.Date(file.lastModified()));
16    }
17 }
```

`lastModified()`方法返回文件最后被修改的日期和时间，它计算的是从UNIX时间（1970年1月1日0时0分0秒）开始的毫秒数，第14~15行使用Date类以一种易读的格式显示它。

图9-16a显示程序在Windows平台上的运行示例，而图9-16b显示程序在UNIX平台上的运行示例。如图9-16所示，Windows平台上和UNIX平台的路径命名习惯是不一样的。



a) 在Windows平台上

b) 在UNIX平台上

图9-16 程序创建一个File对象然后显示文件属性

## 9.7 文件输入和输出

File对象封装了文件或路径的属性，但是它既不包括创建文件，也不包括从（向）文件读（写）数据的方法。为了完成I/O操作，需要使用恰当的Java I/O类创建对象。这些对象包含从（向）文件读（写）数据的方法。本节介绍如何使用Scanner和PrintWriter类从（向）文本文件读（写）字符串和数值信息。



### 9.7.1 使用PrintWriter写数据

`java.io.PrintWriter`类可用来创建一个文件并向文本文件写入数据。首先，必须为一个文本文件创建一个`PrintWriter`对象，如下所示：

```
PrintWriter output = new PrintWriter(filename);
```

然后，可以调用`PrintWriter`对象上的`print`、`println`和`printf`方法向文件写入数据。图9-17总结了`PrintWriter`中的常用方法。

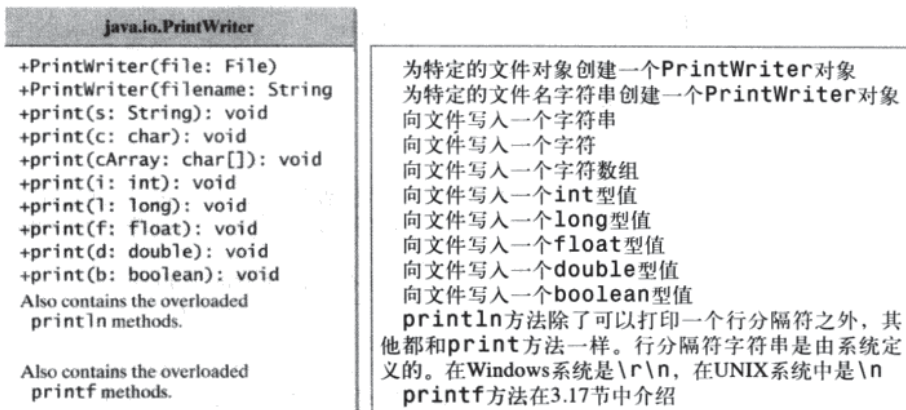


图9-17 `PrintWriter`类包括将数据写入文本文件的方法

程序清单9-7给出创建一个`PrintWriter`实例并且向文件“score.txt”中写入两行数据的例子。每行都包括名字（字符串）、中间名字的首字母（字符）、姓（字符串）和分数（整数）。

程序清单9-7 WriteData.java

```
1 public class WriteData {
2     public static void main(String[] args) throws Exception {
3         java.io.File file = new java.io.File("scores.txt");
4         if (file.exists()) {
5             System.out.println("File already exists");
6             System.exit(0);
7         }
8
9         // Create a file
10        java.io.PrintWriter output = new java.io.PrintWriter(file);
11
12        // Write formatted output to the file
13        output.print("John T Smith ");
14        output.println(90);
15        output.print("Eric K Jones ");
16        output.println(85);
17
18        // Close the file
19        output.close();
20    }
21 }
```

第3~7行检查文件score.txt是否存在。如果存在，则退出该程序（第6行）。

如果文件不存在，调用`PrintWriter`的构造方法会创建一个新文件。如果文件已经存在，那么文件的当前内容将被废弃。

调用`PrintWriter`的构造方法可能会抛出某种I/O异常。Java强制要求编写代码来处理这类异常。在13章中将学习如何处理它。目前，只要在方法头声明中声明`throws Exception`（第2行）即可。

你已经使用过`System.out.print`和`System.out.println`方法向控制台输出文本。`System.out`是控制台的标准Java对象。可以创建对象，然后使用`print`、`println`和`printf`向文件中写入文本（第

13~16行)。

必须使用`close()`方法关闭文件。如果没有调用该方法,数据就不能正确地保存在文件中。

### 9.7.2 使用Scanner读数据

在2.3节中,`java.util.Scanner`类用来从控制台读取字符串和基本类型数值。`Scanner`可以将输入分为由空白字符分隔的有用信息。为了能从键盘读取,需要为`System.in`创建一个`Scanner`,如下所示:

```
Scanner input = new Scanner(System.in);
```

为了从文件中读取,为文件创建一个`Scanner`,如下所示:

```
Scanner input = new Scanner(new File(filename));
```

图9-18总结出`Scanner`中的常用方法。

java.util.Scanner	
+Scanner(source: File)	创建一个所产生的值都是从特定文件扫描而来的扫描器
+Scanner(source: String)	创建一个所产生的值都是从特定字符串扫描而来的扫描器
+close()	关闭这个扫描器
+hasNext(): boolean	如果这个扫描器还有可读的数据则返回true
+next(): String	从这个扫描器返回下一个标志作为字符串
+nextLine(): String	使用行分隔符从这个扫描器返回一个行结束
+nextByte(): byte	从这个扫描器返回下一个标志作为一个byte值
+nextShort(): short	从这个扫描器返回下一个标志作为一个short值
+nextInt(): int	从这个扫描器返回下一个标志作为一个int值
+nextLong(): long	从这个扫描器返回下一个标志作为一个long值
+nextFloat(): float	从这个扫描器返回下一个标志作为一个float值
+nextDouble(): double	从这个扫描器返回下一个标志作为一个double值
+useDelimiter(pattern: String): Scanner	设置这个扫描器的分隔模式并返回这个扫描器

图9-18 Scanner类包含扫描数据的方法

程序清单9-8给出的例子创建了一个`Scanner`的实例,并从文件“scores.txt”中读取数据。

#### 程序清单9-8 ReadData.java

```
1 import java.util.Scanner;
2
3 public class ReadData {
4     public static void main(String[] args) throws Exception {
5         // Create a File instance
6         java.io.File file = new java.io.File("scores.txt");
7
8         // Create a Scanner for the file
9         Scanner input = new Scanner(file);
10
11        // Read data from a file
12        while (input.hasNext()) {
13            String firstName = input.next();
14            String mi = input.next();
15            String lastName = input.next();
16            int score = input.nextInt();
17            System.out.println(
18                firstName + " " + mi + " " + lastName + " " + score);
19        }
20
21        // Close the file
22        input.close();
23    }
24 }
```

scores.txt

John	T	Smith	90
Eric	K	Jones	85

注意, `new Scanner(String)`为给定的字符串创建一个`Scanner`。为创建`Scanner`从文件中读取数据,必须使用构造方法`new File(filename)`利用`java.io.File`类创建`File`的一个实例(第6行),

然后使用`new Scanner(File)`为文件创建一个`Scanner` (第9行)。

调用构造方法`new Scanner(File)`可能会抛出一个I/O异常。因此, `main`方法在第4行声明了`throws Exception`。

`while`循环中的每次迭代都从文本文件中读取名字、中间名、姓和分数 (第12~19行)。文件在第22行关闭。

没有必要关闭输入文件 (第22行), 但这样做是一种释放被文件占用的资源的好方法。

### 9.7.3 Scanner如何工作

方法`nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`、`nextDouble()`和`next()`等都称为令牌读取方法 (token-reading method), 因为它们会读取用分隔符分隔开的令牌。默认情况下, 分隔符是空格。可以使用`useDelimiter(String regex)`方法设置新的分隔符模式。

一个输入方法是如何工作的呢? 一个令牌读取方法首先跳过任意分隔符 (默认情况下是空格), 然后读取一个以分隔符结束的令牌。然后, 对应于`nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`和`nextDouble()`, 这个令牌就分别被自动地转换为一个`byte`、`short`、`int`、`long`、`float`或`double`型的值。对于`next()`方法而言是无须做转换的。如果令牌和期望的类型不匹配, 就会抛出一个运行异常`java.util.InputMismatchException`。

方法`next()`和`nextLine()`都会读取一个字符串。`next()`方法读取一个由分隔符分隔的字符串, 但是`nextLine()`读取一个以行分隔符结束的行。

**注意** 行分隔符字符串是由系统定义的, 在Windows平台上是`\r\n`, 而在UNIX平台上是`\n`。

为了得到特定平台上的行分隔符, 使用

```
String lineSeparator = System.getProperty("line.separator");
```

如果从键盘输入, 每行就以回车键 (Enter key) 结束, 它对应于`\n`字符。

令牌读取方法不能读取令牌后面的分隔符。如果在令牌读取方法之后调用`nextLine()`, 该方法读取从这个分隔符开始, 到这行的行分隔符结束的字符。这个行分隔符也被读取, 但是它不是`nextLine()`返回的字符串部分。

假设一个名为`test.txt`的文本文件包含一行

```
34 567
```

在执行完下面的代码之后,

```
Scanner input = new Scanner(new File("test.txt"));
int intValue = input.nextInt();
String line = input.nextLine();
```

`intValue`的值为34, 而`line`包含的字符是' '、'5'、'6'、'7'。

如果输入是从键盘键入, 那会发生什么呢? 假设为下面的代码输入34, 然后按回车键, 接着输入567, 然后再按回车键:

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
String line = input.nextLine();
```

将会得到`intValue`值是34, 而`line`中是一个空的字符串。这是为什么呢? 原因如下。令牌读取方法`nextInt()`读取34, 然后在分隔符处停止, 这里的分隔符是行分隔符 (回车键)。`nextLine()`方法会在读取行分隔符之后结束, 然后返回在行分隔符之前的字符串。因为在行分隔符之前没有字符, 所以`line`是空的。

### 9.7.4 问题: 替换文本

假设要编写一个名为`ReplaceText`的程序, 用一个新字符串替换文本文件中所有出现某个字符串的地方。文件名和字符串都作为命令行参数传递, 如下所示:

```
java ReplaceText sourceFile targetFile oldString newString
```

例如, 调用

```
java ReplaceText FormatString.java t.txt StringBuilder StringBuffer
```

就会用StringBuffer替换FormatString.java中所有出现的StringBuilder, 然后将新文件保存在t.txt中。

程序清单9-9给出这个问题的解决方案。程序检查传给main方法的参数个数 (第7~11行), 检查源文件和目标文件是否存在 (第14~25行), 为源文件创建一个Scanner (第28行), 为目标文件创建一个PrintWriter, 然后重复从源文件读入一行 (第32行), 替换文本 (第33行), 向目标文件中写入新的一行 (第34行)。必须关闭输出文件 (第38行), 以确保数据正确地保存在文件中。

**程序清单9-9 ReplaceText.java**

```

1 import java.io.*;
2 import java.util.*;
3
4 public class ReplaceText {
5     public static void main(String[] args) throws Exception {
6         // Check command-line parameter usage
7         if (args.length != 4) {
8             System.out.println(
9                 "Usage: java ReplaceText sourceFile targetFile oldStr newStr");
10            System.exit(0);
11        }
12
13        // Check if source file exists
14        File sourceFile = new File(args[0]);
15        if (!sourceFile.exists()) {
16            System.out.println("Source file " + args[0] + " does not exist");
17            System.exit(0);
18        }
19
20        // Check if target file exists
21        File targetFile = new File(args[1]);
22        if (targetFile.exists()) {
23            System.out.println("Target file " + args[1] + " already exists");
24            System.exit(0);
25        }
26
27        // Create a Scanner for input and a PrintWriter for output
28        Scanner input = new Scanner(sourceFile);
29        PrintWriter output = new PrintWriter(targetFile);
30
31        while (input.hasNext()) {
32            String s1 = input.nextLine();
33            String s2 = s1.replaceAll(args[2], args[3]);
34            output.println(s2);
35        }
36
37        input.close();
38        output.close();
39    }
40 }
```

## 9.8 (GUI) 文件对话框

Java提供javax.swing.JFileChooser类来显示文件对话框, 如图9-19所示。在这个对话框中, 用户可以选择一个文件。

程序清单9-10给出的程序提示用户选择一个文件, 然后在控制台上显示它的内容。



## 程序清单9-10 ReadFileUsingJFileChooser.java

```

1 import java.util.Scanner;
2 import javax.swing.JFileChooser;
3
4 public class ReadFileUsingJFileChooser {
5     public static void main(String[] args) throws Exception {
6         JFileChooser fileChooser = new JFileChooser();
7         if (fileChooser.showOpenDialog(null)
8             == JFileChooser.APPROVE_OPTION) {
9             // Get the selected file
10            java.io.File file = fileChooser.getSelectedFile();
11
12            // Create a Scanner for the file
13            Scanner input = new Scanner(file);
14
15            // Read text from the file
16            while (input.hasNext()) {
17                System.out.println(input.nextLine());
18            }
19
20            // Close the file
21            input.close();
22        }
23        else {
24            System.out.println("No file selected");
25        }
26    }
27 }

```

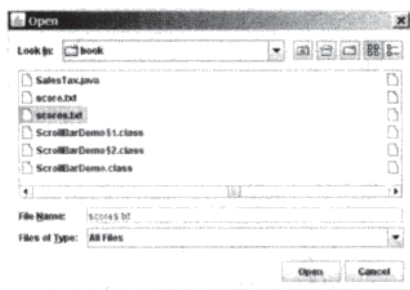


图9-19 JFileChooser可以用来显示一个打开某文件的文件对话框

程序在第6行创建一个JFileChooser。showOpenDialog(null)方法显示一个对话框，如图9-19所示。这个方法返回一个int型值，或者是APPROVE\_OPTION或者是CANCEL\_OPTION，它们分别表明点击Open按钮或者点击Cancel按钮。

getSelectedFile()方法（第10行）返回从文件对话框中选中的文件。第13行创建该文件的一个scanner。程序不断地从文件读取一行，然后将它们显示在控制台（第16~18行）。

## 本章小结

- 字符串是封装在String类中的对象。要创建一个字符串，可以使用11种构造方法之一，也可以使用字符串直接量进行简洁初始化。
- String对象是不可变的，它的内容不能改变。为了提高效率和节省内存，如果两个直接量字符串有相同的字符序列，Java虚拟机就将它们存储在一个对象中。这个独特的对象称为限定字符串对象。
- 可以调用字符串的length()方法获取它的长度，使用charAt(index)方法从字符串中提取特定下标位置的字符，使用indexOf和lastIndexOf方法找出一个字符串中的某个字符或某个子串。

- 可以使用concat方法连接两个字符串，或者使用加号(+)连接两个或两个以上的字符串。
- 可以使用substring方法从字符串中提取子串。
- 可以使用equals和compareTo方法比较字符串。如果两个字符串相等，equals方法返回true，如果它们不等，则返回false。compareTo方法根据一个字符串等于、大于或小于另一个字符串，分别返回0、正整数或负整数。
- Character类是单个字符的包装类。Character类提供很多实用的静态方法，用于判断一个字符是否是字母(isLetter(char))、数字(isDigit(char))、大写字母(isUpperCase(char))或小写字母(isLowerCase(char))。
- StringBuilder/StringBuffer类可以用来替代String类。String对象是不可改变的，但是可以向StringBuilder/StringBuffer对象中添加、插入或追加新的内容。如果字符串的内容不需要任何改变，就使用String类；如果需要改变，则使用StringBuilder/StringBuffer类。
- 可以从命令行向main方法传递字符串。传递给main程序的字符串存储在一个字符串数组args中。第一个字符串用args[0]表示，而arg.length表示传入的字符串的个数。
- File类用来获取文件的属性和对文件进行操作。它不包括创建文件，以及从(向)文件读(写)数据的方法。
- 可以使用Scanner从文本文件中读取字符串和基本类型数据值，使用PrintWriter创建一个文件并向文本文件中写入数据。
- 可以使用JFileChooser类以图形的形式显示文件。

## 复习题

### 9.2节

9.1 假设s1、s2、s3和s4是四个字符串，如下给出其值：

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
String s4 = "Welcome to Java";
```

那么下面表达式的结果是什么？

- |                              |                                |
|------------------------------|--------------------------------|
| (1) s1 == s2                 | (13) s1.length()               |
| (2) s2 == s3                 | (14) s1.substring(5)           |
| (3) s1.equals(s2)            | (15) s1.substring(5, 11)       |
| (4) s2.equals(s3)            | (16) s1.startsWith("Wel")      |
| (5) s1.compareTo(s2)         | (17) s1.endsWith("Java")       |
| (6) s2.compareTo(s3)         | (18) s1.toLowerCase()          |
| (7) s1 == s4                 | (19) s1.toUpperCase()          |
| (8) s1.charAt(0)             | (20) " Welcome ".trim()        |
| (9) s1.indexOf('j')          | (21) s1.replace('o', 'T')      |
| (10) s1.indexOf("to")        | (22) s1.replaceAll("o", "T")   |
| (11) s1.lastIndexOf('a')     | (23) s1.replaceFirst("o", "T") |
| (12) s1.lastIndexOf("o", 15) | (24) s1.toCharArray()          |

为了创建一个字符串“Welcome to Java”，可能会用到如下所示的语句：

```
String s = "Welcome to Java";
```

或者

```
String s = new String("Welcome to Java");
```

哪个语句更好？为什么？

9.2 假设s1和s2是两个字符串，下面哪些语句或表达式是错误的？

```
String s = new String("new string");
String s3 = s1 + s2;
String s3 = s1 - s2;
```

```

s1 == s2;
s1 >= s2;
s1.compareTo(s2);
int i = s1.length();
char c = s1(0);
char c = s1.charAt(s1.length());

```

9.3 下面代码的输出是什么?

```

String s1 = "Welcome to Java";
String s2 = s1.replace("o", "abc");
System.out.println(s1);
System.out.println(s2);

```

9.4 假设s1是"Welcome"而s2是"welcome"为下面的陈述编写代码:

- 检查s1和s2是否相等,然后将结果赋值给一个布尔变量isEqual。
- 检查在忽略大小写的情况下s1和s2是否相等,然后将结果赋值给一个布尔变量isEqual。
- 比较s1和s2,然后将结果赋值给一个整型值x。
- 在忽略大小写的情况下比较s1和s2,然后将结果赋值给一个整型值x。
- 检查s1是否有前缀"AAA",然后将结果赋值给一个布尔变量b。
- 检查s1是否有后缀"AAA",然后将结果赋值给一个布尔变量b。
- 将s1的长度赋值给一个整型变量x。
- 将s1的第一个字符赋值给一个字符型变量x。
- 创建一个新字符串s3,它是s1和s2的组合。
- 创建一个s1的子串,下标从1开始。
- 创建一个s1的子串,下标从1到4。
- 创建一个新字符串s3,它将s1转换为小写。
- 创建一个新字符串s3,它将s1转换为大写。
- 创建一个新字符串s3,它将s1的两端的空格去掉。
- 用E替换s1中所有出现字符e的地方,然后将新字符串赋值给s3。
- 将"Welcome to Java and HTML"按空格分隔为一个数组tokens。
- 将s1中字符e第一次出现的下标赋值给一个int型变量x。
- 将s1中字符串abc最后一次出现的下标赋值给一个int型变量x。

9.5 String类中是否有可以改变字符串内容的方法?

9.6 假设字符串s是用new String()创建的,那么s.length()是多少?

9.7 如何将一个char值、一个字符数组或一个数值转换为一个字符串?

9.8 为什么下面的代码会造成NullPointerException异常?

```

1 public class Test {
2     private String text;
3
4     public Test(String s) {
5         String text = s;
6     }
7
8     public static void main(String[] args) {
9         Test test = new Test("ABC");
10        System.out.println(test.text.toLowerCase());
11    }
12 }

```

9.9 下面程序的错误是什么?

```

1 public class Test {
2     String text;
3
4     public void Test(String s) {

```



```

5     this.text = s;
6 }
7
8 public static void main(String[] args) {
9     Test test = new Test("ABC");
10    System.out.println(test);
11 }
12 }

```

### 9.3节

9.10 怎样判断一个字母是大写还是小写?

9.11 怎样判断一个字符是否为字母或数字?

### 9.4节

9.12 `StringBuilder`和`StringBuffer`之间的区别是什么?

9.13 如何为一个字符串创建字符串生成器? 如何从一个字符串生成器获取字符串?

9.14 使用`StringBuilder`类中的`reverse`方法编写三条语句, 倒置字符串`s`。

9.15 编写一条语句, 从包含20个字符的字符串`s`中删除下标从4到10的子串。使用`StringBuilder`类中的`delete`方法。

9.16 字符串和字符串生成器的内部结构是什么?

9.17 假设给出如下所示的`s1`和`s2`:

```

StringBuilder s1 = new StringBuilder("Java");
StringBuilder s2 = new StringBuilder("HTML");

```

显示执行每条语句之后`s1`的结果。假定这些表达式都是相互独立的。

(1) <code>s1.append(" is fun");</code>	(7) <code>s1.deleteCharAt(3);</code>
(2) <code>s1.append(s2);</code>	(8) <code>s1.delete(1, 3);</code>
(3) <code>s1.insert(2, "is fun");</code>	(9) <code>s1.reverse();</code>
(4) <code>s1.insert(1, s2);</code>	(10) <code>s1.replace(1, 3, "Computer");</code>
(5) <code>s1.charAt(2);</code>	(11) <code>s1.substring(1, 3);</code>
(6) <code>s1.length();</code>	(12) <code>s1.substring(2);</code>

9.18 给出下面程序的输出结果:

```

public class Test {
    public static void main(String[] args) {
        String s = "Java";
        StringBuilder builder = new StringBuilder(s);
        change(s, builder);

        System.out.println(s);
        System.out.println(builder);
    }

    private static void change(String s, StringBuilder builder) {
        s = s + " and HTML";
        builder.append(" and HTML");
    }
}

```

### 9.5节

9.19 本书声明`main`方法为:

```
public static void main(String[] args)
```

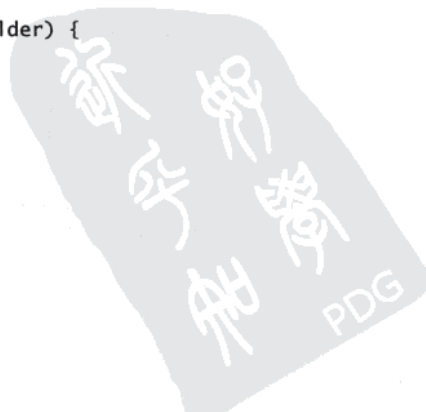
能将它替换成下面的某条语句吗?

```
public static void main(String args[])
```

```
public static void main(String[] x)
```

```
public static void main(String x[])
```

```
static void main(String x[])
```





## 9.20 显示当使用命令

- (1) `java Test I have a dream`
- (2) `java Test "1 2 3"`
- (3) `java Test`
- (4) `java Test "*"`
- (5) `java Test *`

调用下面的程序时，程序的输出结果。

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Number of strings is " + args.length);
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

## 9.6节

## 9.21 使用下面的语句创建File对象时，错在哪里？

```
new File("c:\book\test.dat");
```

## 9.22 如何检查一个文件是否已经存在？如何删除一个文件？如何重命名一个文件？使用File类能够获得文件的大小（字节数）吗？

## 9.23 能够使用File类进行输入/输出吗？创建一个File对象就是在磁盘上创建一个文件吗？

## 9.7节

## 9.24 如何创建一个PrintWriter向文件写数据？在程序清单9-7中，为什么要在main方法中声明throws Exception？在程序清单9-7中，如果不调用close()方法，将会发生什么？

## 9.25 给出下面的程序执行之后，文件temp.txt的内容。

```
public class Test {
    public static void main(String[] args) throws Exception {
        java.io.PrintWriter output = new
            java.io.PrintWriter("temp.txt");
        output.printf("amount is %f %e\r\n", 32.32, 32.32);
        output.printf("amount is %5.4f %5.4e\r\n", 32.32, 32.32);
        output.printf("%6b\r\n", (1 > 2));
        output.printf("%6s\r\n", "Java");
        output.close();
    }
}
```

## 9.26 如何创建一个Scanner从文件读数据？在程序清单9-8中，为什么要在main方法中声明throws Exception？在程序清单9-8中，如果不调用close()方法，将会发生什么？

## 9.27 如果试图对一个不存在的文件创建Scanner，将会发生什么情况？如果试图对一个已经存在的文件创建PrintWriter，会发生什么情况？

## 9.28 是否所有平台上的行分隔符都是一样的？Windows平台上的行分隔符是什么？

## 9.29 假设输入45 57.8 789，然后点击回车键。显示执行完下面的代码之后变量的内容。

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
double doubleValue = input.nextDouble();
String line = input.nextLine();
```

## 9.30 假设输入45、回车键、57.8、回车键、789、回车键。显示执行完下面的代码之后变量的内容。

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
double doubleValue = input.nextDouble();
String line = input.nextLine();
```

## 编程练习题

### 9.2~9.3节

\*9.1 (检查SSN) 编写程序, 提示用户输入一个社会保险号码, 它的格式是DDD-DD-DDDD, 其中D是一个数字。程序会为正确的社保号显示"Valid SSN"; 否则, 显示"Invalid SSN"。

\*\*9.2 (检测子串) 可以使用String类中的indexOf方法检测一个字符串是否是另一个字符串的子串。编写自己的方法实现这个功能。编写一个程序, 提示用户输入两个字符串, 检测第一个字符串是否是第二个字符串的子串。

\*\*9.3 (检验密码) 一些网站设定了一些制定密码的规则。编写一个方法, 检验一个字符串是否是合法的密码。假设密码规则如下:

- 密码必须至少有8个字符。
- 密码只能包括字母和数字。
- 密码必须至少有2个数字。

编写一个程序, 提示用户输入密码, 如果该密码符合规则就显示"Valid Password", 否则显示"Invalid Password"。

9.4 (求某个指定字符的出现次数) 使用下面的方法头编写一个方法, 找出某个指定字符在字符串中出现的次数:

```
public static int count(String str, char a)
```

例如, count("Welcome", 'e') 返回2。编写一个测试程序, 提示用户输入一个字符串, 在该字符串后紧跟着一个字符, 然后显示这个字符在字符串中出现的次数。

\*\*9.5 (在字符串中每个数字出现的次数) 使用下面的方法头编写一个方法, 统计每个数字在字符串中出现的次数:

```
public static int[] count(String s)
```

这个方法统计每个数字在字符串中出现的次数。返回值是10个元素构成的数组, 每个元素存储的是一个数字出现的次数。例如, 在执行完int[] counts = count("12203AB3")之后, counts[0]为1, counts[1]为1, counts[2]为2, counts[3]为2。

编写一个测试程序, 提示用户输入一个字符串, 然后显示每个数字在字符串中出现的次数。

\*9.6 (统计字符串中字母的个数) 使用下面的方法头编写一个方法, 统计字母在字符串中出现的个数。

```
public static int countLetters(String s)
```

编写一个测试程序, 提示用户输入一个字符串, 然后显示这个字符串中字母的个数。

\*9.7 (电话按键盘) 国际标准的字母/数字匹配图可以在电话上找到, 如下所示:

1	2	3
	ABC	DEF
4	5	6
GHI	JKL	MNO
7	8	9
PQRS	TUV	WXYZ
	0	

编写一个方法返回给定大写字母的数字, 如下所示:

```
public static int getNumber(char uppercaseLetter)
```

编写一个测试程序, 提示用户输入字符串形式的电话号码。输入的数字可能会包含字母。程序将字母(大写或者小写)翻译成一个数字, 然后保持其他字符不动。下面是该程序的运行示例:

Enter a string: 1-800-Flowers  
1-800-3569377



Enter a string: 1800flowers  
18003569377



**\*9.8 (二进制转换为十进制)** 编写一个方法，将一个二进制数字作为一个字符串转换为一个十进制整数。这个方法头如下所示：

```
public static int binaryToDecimal(String binaryString)
```

例如，二进制字符串10001表示17 ( $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 = 17$ )。所以，`binaryToDecimal("10001")`返回的是17。注意，`Integer.parseInt("10001", 2)`也可以将一个二进制字符串转变成十进制的数值。在这个练习中不要使用该方法。

编写一个测试程序，提示用户输入一个二进制数，然后显示对应的十进制整数值。

#### 9.4节

**\*\*9.9 (将二进制数转换为十六进制数)** 编写一个方法，将一个二进制数转换为一个十六进制数。方法头如下所示：

```
public static String binaryToHex(String binaryValue)
```

编写一个测试程序，提示用户输入一个二进制数，然后显示对应的十六进制数。

**\*\*9.10 (将十进制数转换为二进制数)** 编写一个方法，将一个十进制数转换为一个二进制数。方法头如下所示：

```
public static String decimalToBinary(int value)
```

编写一个测试程序，提示用户输入一个十进制整数，然后显示对应的二进制数。

**\*\*9.11 (对字符串中的字符排序)** 使用下面的方法头编写一个方法，返回排好序的字符串

```
public static String sort(String s)
```

例如，`sort("acb")`返回abc。

编写一个测试程序，提示用户输入一个字符串，然后显示排好序的字符串。

**\*\*9.12 (变位词)** 编写一个方法，检测两个单词是否互为变位词。如果在不计顺序的情况下两个单词包含完全相同的字母，则称这两个单词互为变位词 (anagram)。例如，"silent"和"listen"互为变位词。该方法的方法头如下所示：

```
public static boolean isAnagram(String s1, String s2)
```

编写一个测试程序，提示用户输入两个字符串，如果它们是变位词，则显示"anagram"，否则显示"not anagram"。

#### 9.5节

**\*9.13 (传递字符串检测回文串)** 改写程序清单9-1，将被检测的字符串以命令行参数的方式传入。

**\*9.14 (求整数的和)** 编写两个程序。第一个程序给main方法传入个数不定的整数，每个整数都是一个独立的字符串，然后显示它们的和。第二个程序给main方法传入个数不定的整数构成的同一个字符串，数字之间被一个空格分隔，然后显示它们的和。将这两个程序分别命名为Exercise9\_14a和Exercise9\_14b，如图9-20所示。

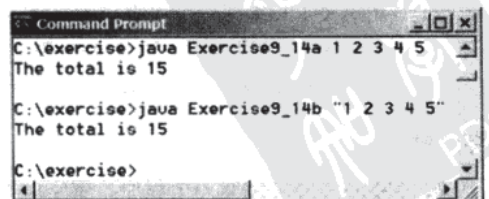


图9-20 程序对从命令行传来的所有数字求和

\*9.15 (求字符串中大写字母的个数) 编写一个程序, 传给main方法一个字符串, 显示该字符串中大写字母的个数。

### 9.7~9.8节

\*\*9.16 (改变Java源代码的格式) 编写一个程序, 将次行块风格的Java源代码转换成行尾块风格。例如, 图a中的Java源代码使用的是次行块风格。程序将它转换成图b中所示的行尾块形式。

```
public class Test
{
    public static void main(String[] args)
    {
        // Some statements
    }
}
```

a) 次行块风格

```
public class Test {
    public static void main(String[] args) {
        // Some statements
    }
}
```

b) 行尾块风格

程序可以从命令行调用, 以Java源代码文件作为其参数。它会将这个Java源代码变成新的格式。例如, 下面的命令将Java源代码文件Test.java转变成行尾块风格:

```
java Exercise9_16 Test.java
```

\*9.17 (统计一个文件中的字符数、单词数和行数) 编写程序统计一个文件中的字符数(控制字符'\r'和'\n'除外)、单词数以及行数。单词由空格、制表符、回车键或换行符分隔, 文件名应该作为命令行参数被传递, 如图9-21所示。

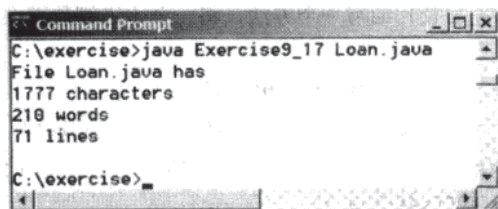


图9-21 程序显示给定文件中的字符数、单词数和行数

\*9.18 (处理文本文件中的分数) 假定一个名为Exercise9\_18.txt的文本文件中包含未指定个数个分数。编写一个程序, 从文件中读入分数, 显示它们的和以及平均值。分数之间用空格分开。

\*9.19 (写/读数据) 编写一个程序, 如果名为Exercise9\_19.txt的文件不存在, 则创建该文件。使用文本I/O编写随机产生100个整数给文件。文件中的整数由空格分开。从文件中读回数据然后显示排好序的数据。

\*\*9.20 (替换文本) 程序清单9-9给出一个程序, 替换源文件中的文本, 然后将这个变化存储到一个新文件中。改写程序, 将这个变化存储到原始文件中。例如:调用

```
java Exercise9_20 file oldString newString
```

用newString代替源文件中的oldString。

\*\*9.21 (删除文本) 编写一个程序, 从一个文本文件中删掉所有出现某个指定字符串的地方。例如, 调用

```
java Exercise9_21 John filename
```

从指定文件中删掉字符串John。

## 综合题

9.22 (猜测首府) 编写一个程序, 重复地提示用户输入一个州的首府, 如图9-22a所示。一旦接收到用户的输入, 程序就会报告这个答案是否正确, 如图9-22b所示。假设有一个二维的数组存储了55个州和它们的首府, 如图9-23所示。程序提示用户回答所有10个州的首府, 然后显示答对的总次数。



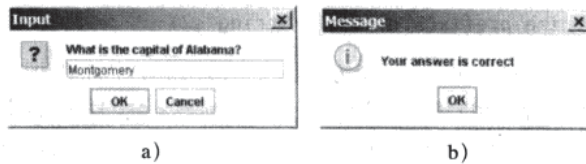


图9-22 程序提示用户在图a中输入首府，然后报告答对的次数

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

图9-23 二维数组存储州名和它们的首府

**\*\*9.23 (实现String类)** Java库中提供了String类。给出你自己对下面方法的实现（将新类命名为MyString1）:

```
public MyString1(char[] chars);
public char charAt(int index);
public int length();
public MyString1 substring(int begin, int end);
public MyString1 toLowerCase();
public boolean equals(MyString1 s);
public static MyString1 valueOf(int i);
```

**\*\*9.24 (实现String类)** 在Java库中提供了String类。给出你自己对下面方法的实现（将新类命名为MyString2）:

```
public MyString2(String s);
public int compare(String s);
public MyString2 substring(int begin);
public MyString2 toUpperCase();
public char[] toChars();
public static MyString2 valueOf(boolean b);
```

**\*\*9.25 (实现Character类)** 在Java库中提供了Character类。给出你自己对这个类的实现。将新类命名为MyCharacter。

**\*\*9.26 (实现StringBuilder类)** 在Java库中提供了StringBuilder类。给出你自己对下面方法的实现（将新类命名为MyStringBuilder1）:

```
public MyStringBuilder1(String s);
public MyStringBuilder1 append(MyStringBuilder1 s);
public MyStringBuilder1 append(int i);
public int length();
public char charAt(int index);
public MyStringBuilder1 toLowerCase();
public MyStringBuilder1 substring(int begin, int end);
public String toString();
```

**\*\*9.27 (实现StringBuilder类)** 在Java库中提供了StringBuilder类。给出你自己对下面方法的实现（将新类命名为MyStringBuilder2）:

```
public MyStringBuilder2();
public MyStringBuilder2(char[] chars);
public MyStringBuilder2(String s);
public MyStringBuilder2 insert(int offset, MyStringBuilder2 s);
public MyStringBuilder2 reverse();
public MyStringBuilder2 substring(int begin);
public MyStringBuilder2 toUpperCase();
```

**\*9.28 (相同的前缀)** 编写一个方法，返回两个字符串共有的前缀。例如，"distance"和"disinfection"的共同前缀是"dis"。方法头如下所示:

```
public static String prefix(String s1, String s2)
```

如果两个字符串没有共同的前缀，这个方法就会返回一个空字符串。

编写一个main方法，提示用户输入两个字符串，然后显示它们共同的前缀。

**\*\*9.29**（新字符串split方法）String类中的split方法会返回一个字符串数组，该数组是由分隔符分隔开的子串构成的。但是，这个分隔符是不返回的。实现下面的新方法，方法返回字符串数组，这个数组由用匹配字符分隔开的子串构成，子串也包括匹配字符。

```
public static String[] split(String s, String regex)
```

例如，split("ab#12#453","#")会返回ab、#、12、#和453构成的String数组，而split("a?b?gf#e","[?#]")会返回a、b、?、b、gf、#和e构成的字符串数组。

**\*\*9.30**（财务问题：信用卡号的合法性）使用信用卡号码作为字符串输入改写练习题5.31。使用下面的方法重新设计这个程序：

```
/** Return true if the card number is valid */
public static boolean isValid(String cardNumber)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(String cardNumber)

/** Return this number if it is a single digit; otherwise,
 * return the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd place digits in number */
public static int sumOfOddPlace(String cardNumber)
```

**\*\*\*9.31**（游戏：刽子手）编写一个刽子手游戏，随机产生一个单词，然后提示用户一次猜测一个字母，如运行示例所示。单词中的每个字母都以星号显示。当用户的猜测准确时，就显示实际的字母。当用户完成了一个单词，就显示猜错的次数，然后询问用户是否继续猜测另一个单词。声明一个数组来存储这些单词，如下所示：

```
// Use any words you wish
String[] words = {"write", "that", ...};
```

```
(Guess) Enter a letter in word ***** > p
(Guess) Enter a letter in word p***** > r
(Guess) Enter a letter in word pr*** > p
      p is already in the word
(Guess) Enter a letter in word pr*** > o
(Guess) Enter a letter in word pro*r** > g
(Guess) Enter a letter in word progr** > n
      n is not in the word
(Guess) Enter a letter in word progr** > m
(Guess) Enter a letter in word progr*m > a
The word is program. You missed 1 time
```

```
Do you want to guess for another word? Enter y or n>
```



**\*\*9.32**（检测ISBN）使用字符串操作来简化练习题3.9。以字符串形式输入一个ISBN的前9个数字。

**\*\*\*9.33**（游戏：刽子手）改写练习题9.31。程序读取存储在一个名为Exercise9\_33.txt的文本文件中的单词。这些单词用空格分隔。

**\*\*9.34**（替换文本）修改练习题Exercise9\_20，使用下面的命令用一个新字符串替换某个特定目录下所有文件中的一个字符串：

```
java Exercise9_34 dir oldString newString
```

\*9.35 (生物信息方面: 找出基因) 生物学家使用字母A、C、T和G构成的序列来对基因组进行建模。基因是基因组的一个子串, 基因组在三字符ATG之后开始, 在三字符TAG、TAA和TGA之前结束。因此, 基因字符串的长度是3的倍数, 而基因不包含任何ATG、TAG、TAA和TGA这样的三字符。

编写一个程序, 提示用户输入一个基因组, 然后显示基因组中所有的基因。如果在输入序列中没有找到任何基因, 就显示无基因。下面是一些运行示例:

```
Enter a genome string: TTATGTTTTAAGGATGGGCGTTAGTT
TTT
GGGCGT
```



```
Enter a genome string: TGTGTGTATAT
no gene is found
```



歡迎  
光臨  
PDG