**Section 14.2 Fundamentals of Characters and Strings**

• A character literal’s value (p. [597](http://proquest.safaribooksonline.com/9780133813036/ch14_html#page_597)) is its integer value in Unicode (p. [597](http://proquest.safaribooksonline.com/9780133813036/ch14_html#page_597)). Strings can include letters, digits and special characters such as +, -, \*, / and $. A string in Java is an object of class String. String literals (p.[597](http://proquest.safaribooksonline.com/9780133813036/ch14_html#page_597)) are often referred to as String objects and are written in a program in double quotes.

#### Section 14.3 Class String

• String objects are immutable (p. [599](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec1_html#page_599))—after they’re created, their character contents cannot be changed.

• String method length (p. [599](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec1_html#page_599)) returns the number of characters in a String.

• String method charAt (p. [599](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec1_html#page_599)) returns the character at a specific position.

• String method regionMatches (p. [601](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec3_html#page_601)) compares portions of two strings for equality.

• String method equals tests for equality. The method returns true if the contents of the Strings are equal,false otherwise. Method equals uses a lexicographical comparison (p. [602](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec3_html#page_602)) for Strings.

• When primitive-type values are compared with ==, the result is true if both values are identical. When references are compared with ==, the result is true if both refer to the same object.

• Java treats all string literals with the same contents as a single String object.

• String method equalsIgnoreCase performs a case-insensitive string comparison.

• String method compareTo uses a lexicographical comparison and returns 0 if the Strings are equal, a negative number if the string that calls compareTo is less than the argument String and a positive number if the string that calls compareTo is greater than than the argument String.

• String methods startsWith and endsWith (p. [604](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec3_html#page_604)) determine whether a string starts with or ends with the specified characters, respectively.

• String method indexOf (p. [605](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec3_html#page_605)) locates the first occurrence of a character or a substring in a string.String method lastIndexOf (p. [605](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec3_html#page_605)) locates the last occurrence of a character or a substring in a string.

• String method substring copies and returns part of an existing string object.

• String method concat (p. [608](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec5_html#page_608)) concatenates two string objects and returns a new string object.

• String method replace returns a new string object that replaces every occurrence in a String of its first character argument with its second character argument.

• String method toUpperCase (p. [609](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec7_html#page_609)) returns a new string with uppercase letters in the positions where the original string had lowercase letters. String methodtoLowerCase (p. [610](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec7_html#page_610)) returns a new string with lowercase letters in the positions where the original string had uppercase letters.

• String method trim (p. [610](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec7_html#page_610)) returns a new string object in which all white-space characters (e.g., spaces, newlines and tabs) have been removed from the beginning and end of a string.

• String method toCharArray (p. [610](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec7_html#page_610)) returns a chararray containing a copy of the string’s characters.

• String class static method valueOf returns its argument converted to a string.

#### Section 14.4 Class StringBuilder

• Class StringBuilder provides constructors that enableStringBuilders to be initialized with no characters and an initial capacity of 16 characters, with no characters and an initial capacity specified in the integer argument, or with a copy of the characters of the String argument and an initial capacity that’s the number of characters in the String argument plus 16.

• StringBuilder method length (p. [612](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec4_html#page_612)) returns the number of characters currently stored in aStringBuilder. StringBuilder method capacity (p.[612](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec4_html#page_612)) returns the number of characters that can be stored in a StringBuilder without allocating more memory.

• StringBuilder method ensureCapacity (p. [613](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec10_html#page_613)) ensures that a StringBuilder has at least the specified capacity. Method setLength increases or decreases the length of a StringBuilder.

• StringBuilder method charAt (p. [614](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec10_html#page_614)) returns the character at the specified index. Method setCharAt (p.[614](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec10_html#page_614)) sets the character at the specified position.StringBuilder method getChars (p. [614](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec10_html#page_614)) copies characters in the StringBuilder into the character array passed as an argument.

• StringBuilder’s overloaded append methods (p. [615](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec12_html#page_615)) add primitive-type, character-array, String, Object orCharSequence (p. [615](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec12_html#page_615)) values to the end of aStringBuilder.

• StringBuilder’s overloaded insert (p. [617](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec12_html#page_617)) methods insert primitive-type, character-array, String, Objector CharSequence values at any position in aStringBuilder.

#### Section 14.5 Class Character

• Character method isDefined (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character is in the Unicode character set.

• Character method isDigit (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character is a defined Unicode digit.

• Character method isJavaIdentifierStart (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character can be used as the first character of a Java identifier. Character methodisJavaIdentifierPart (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines if a character can be used in an identifier.

• Character method isLetter (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character is a letter. Character methodisLetterOrDigit (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines if a character is a letter or a digit.

• Character method isLowerCase (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character is a lowercase letter. Charactermethod isUpperCase (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) determines whether a character is an uppercase letter.

• Character method toUpperCase (p. [620](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_620)) converts a character to its uppercase equivalent. Charactermethod toLowerCase (p. [621](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_621)) converts a character to its lowercase equivalent.

• Character method digit (p. [621](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_621)) converts its character argument into an integer in the number system specified by its integer argument radix (p.[621](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_621)). Character method forDigit (p. [621](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_621)) converts its integer argument digit into a character in the number system specified by its integer argument radix.

• Character method charValue (p. [622](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec5_html#page_622)) returns thechar stored in a Character object. Character methodtoString returns a String representation of aCharacter.

#### Section 14.6 Tokenizing Strings

• Class String’s split method (p. [623](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec6_html#page_623)) tokenizes aString based on the delimiter (p. [623](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec6_html#page_623)) specified as an argument and returns an array of Strings containing the tokens (p. [623](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec6_html#page_623)).

#### Section 14.7 Regular Expressions, Class Patternand Class Matcher

• Regular expressions (p. [624](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec7_html#page_624)) are sequences of characters and symbols that define a set of strings. They’re useful for validating input and ensuring that data is in a particular format.

• String method matches (p. [624](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec7_html#page_624)) receives a string that specifies the regular expression and matches the contents of the String object on which it’s called to the regular expression. The method returns a booleanindicating whether the match succeeded.

• A character class is an escape sequence that represents a group of characters. Each character class matches a single character in the string we’re attempting to match with the regular expression.

• A word character (\w; p. [624](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec7_html#page_624)) is any letter (uppercase or lowercase), any digit or the underscore character.

• A white-space character (\s) is a space, a tab, a carriage return, a newline or a form feed.

• A digit (\d) is any numeric character.

• To match a set of characters that does not have a predefined character class (p. [624](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec7_html#page_624)), use square brackets, []. Ranges can be represented by placing a dash (-) between two characters. If the first character in the brackets is "^", the expression accepts any character other than those indicated.

• When the regular expression operator "\*" appears in a regular expression, the program attempts to match zero or more occurrences of the subexpression immediately preceding the "\*".

• Operator "+" attempts to match one or more occurrences of the subexpression preceding it.

• The character "|" allows a match of the expression to its left or to its right.

• Parentheses () are used to group parts of the regular expression.

• The asterisk (\*) and plus (+) are formally called quantifiers (p. [628](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec7_html#page_628)).

• A quantifier affects only the subexpression immediately preceding it.

• Quantifier question mark (?) matches zero or one occurrences of the expression that it quantifies.

• A set of braces containing one number ({n}) matches exactly n occurrences of the expression it quantifies. Including a comma after the number enclosed in braces matches at least n occurrences.

• A set of braces containing two numbers ({n,m}) matches between n and m occurrences of the expression that it qualifies.

• Quantifiers are greedy (p. [629](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec14_html#page_629))—they’ll match as many occurrences as they can as long as the match is successful. If a quantifier is followed by a question mark (?), the quantifier becomes reluctant (p. [629](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec14_html#page_629)), matching as few occurrences as possible as long as the match is successful.

• String method replaceAll (p. [629](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec14_html#page_629)) replaces text in a string with new text (the second argument) wherever the original string matches a regular expression (the first argument).

• Escaping a special regular-expression character with a\ instructs the regular-expression matching engine to find the actual character, as opposed to what it represents in a regular expression.

• String method replaceFirst (p. [629](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec14_html#page_629)) replaces the first occurrence of a pattern match and returns a new string in which the appropriate characters have been replaced.

• String method split (p. [629](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec14_html#page_629)) divides a string into substrings at any location that matches a specified regular expression and returns an array of the substrings.

• Class Pattern (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) represents a regular expression.

• Class Matcher (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) contains a regular-expression pattern and a CharSequence in which to search.

• CharSequence is an interface (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) that allows read access to a sequence of characters. Both String andStringBuilder implement this interface, so they can be used with class Matcher.

• If a regular expression will be used only once, staticPattern method matches (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) takes a string that specifies the regular expression and a CharSequence on which to perform the match. This method returns aboolean indicating whether the search object matches the regular expression.

• If a regular expression will be used more than once, it’s more efficient to use static Pattern method compile (p.[631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) to create a specific Pattern object for that regular expression. This method receives a string representing the pattern and returns a new Pattern object.

• Pattern method matcher (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) receives aCharSequence to search and returns a Matcher object.Matcher method matches (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) performs the same task as Pattern method matches but without arguments.

• Matcher method find (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) attempts to match a piece of the search object to the search pattern. Each call to this method starts at the point where the last call ended, so multiple matches can be found.

• Matcher method lookingAt (p. [631](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec15_html#page_631)) performs the same as find, except that it always starts from the beginning of the search object and will always find the first match if there is one.

• Matcher method group (p. [632](http://proquest.safaribooksonline.com/9780133813036/ch14lev2sec16_html#page_632)) returns the string from the search object that matches the search pattern. The string returned is the one that was last matched by a call to find or lookingAt.

### Self-Review Exercises

[**14.1**](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec11_html#ch14ans1) State whether each of the following is true or false. Iffalse, explain why.

a) When String objects are compared using ==, the result is true if the Strings contain the same values.

b) A String can be modified after it’s created.

[**14.2**](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec11_html#ch14ans2) For each of the following, write a single statement that performs the indicated task:

a) Compare the string in s1 to the string in s2 for equality of contents.

b) Append the string s2 to the string s1, using +=.

c) Determine the length of the string in s1.

### Answers to Self-Review Exercises

[**14.1**](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec10_html#ch14que1)

a) False. String objects are compared using operator == to determine whether they’re the same object in memory.

b) False. String objects are immutable and cannot be modified after they’re created. StringBuilder objects can be modified after they’re created.

[**14.2**](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec10_html#ch14que2)

a) s1.equals(s2)

b) s1 += s2;

c) s1.length()

### Exercises

**14.3 (Comparing** ***String*s)** Write an application that usesString method compareTo to compare two strings input by the user. Output whether the first string is less than, equal to or greater than the second.

**14.4 (Comparing Portions of** ***String*s)** Write an application that uses String method regionMatches to compare two strings input by the user. The application should input the number of characters to be compared and the starting index of the comparison. The application should state whether the strings are equal. Ignore the case of the characters when performing the comparison.

**14.5 (Random Sentences)** Write an application that uses random-number generation to create sentences. Use four arrays of strings called article, noun, verb andpreposition. Create a sentence by selecting a word at random from each array in the following order: article,noun, verb, preposition, article and noun. As each word is picked, concatenate it to the previous words in the sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The application should generate and display 20 sentences.

The article array should contain the articles "the", "a","one", "some" and "any"; the noun array should contain the nouns "boy", "girl", "dog", "town" and "car"; the verb array should contain the verbs "drove", "jumped", "ran","walked" and "skipped"; the preposition array should contain the prepositions "to", "from", "over", "under" and"on".

**14.6 (Project: Limericks)** A limerick is a humorous five-line verse in which the first and second lines rhyme with the fifth, and the third line rhymes with the fourth. Using techniques similar to those developed in [Exercise 14.5](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec12_html#ch14que5), write a Java application that produces random limericks. Polishing this application to produce good limericks is a challenging problem, but the result will be worth the effort!

**14.7 (Pig Latin)** Write an application that encodes English-language phrases into pig Latin. Pig Latin is a form of coded language. There are many different ways to form pig Latin phrases. For simplicity, use the following algorithm:

To form a pig Latin phrase from an English-language phrase, tokenize the phrase into words with String methodsplit. To translate each English word into a pig Latin word, place the first letter of the English word at the end of the word and add the letters “ay.” Thus, the word “jump” becomes “umpjay,” the word “the” becomes “hetay,” and the word “computer” becomes “omputercay.” Blanks between words remain as blanks. Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks and all words have two or more letters. Method printLatinWord should display each word. Each token is passed to method printLatinWord to print the pig Latin word. Enable the user to input the sentence. Keep a running display of all the converted sentences in a text area.

**14.8 (Tokenizing Telephone Numbers)** Write an application that inputs a telephone number as a string in the form (555) 555-5555. The application should use Stringmethod split to extract the area code as a token, the first three digits of the phone number as a token and the last four digits of the phone number as a token. The seven digits of the phone number should be concatenated into one string. Both the area code and the phone number should be printed. Remember that you’ll have to change delimiter characters during the tokenization process.

**14.9 (Displaying a Sentence with Its Words Reversed)**Write an application that inputs a line of text, tokenizes the line with String method split and outputs the tokens in reverse order. Use space characters as delimiters.

**14.10 (Displaying** ***String*s in Uppercase and Lowercase)** Write an application that inputs a line of text and outputs the text twice—once in all uppercase letters and once in all lowercase letters.

**14.11 (Searching** ***String*s)** Write an application that inputs a line of text and a search character and uses Stringmethod indexOf to determine the number of occurrences of the character in the text.

**14.12 (Searching** ***String*s)** Write an application based on the application in [Exercise 14.11](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec12_html#ch14que11) that inputs a line of text and uses String method indexOf to determine the total number of occurrences of each letter of the alphabet in the text. Uppercase and lowercase letters should be counted together. Store the totals for each letter in an array, and print the values in tabular format after the totals have been determined.

**14.13 (Tokenizing and Comparing** ***String*s)** Write an application that reads a line of text, tokenizes the line using space characters as delimiters and outputs only those words beginning with the letter "b".

**14.14 (Tokenizing and Comparing** ***String*s)** Write an application that reads a line of text, tokenizes it using space characters as delimiters and outputs only those words ending with the letters "ED".

**14.15 (Converting** ***int*** **Values to Characters)** Write an application that inputs an integer code for a character and displays the corresponding character. Modify this application so that it generates all possible three-digit codes in the range from 000 to 255 and attempts to print the corresponding characters.

**14.16 (Defining Your Own** ***String*** **Methods)** Write your own versions of String search methods indexOf andlastIndexOf.

**14.17 (Creating Three-Letter** ***String*s from a Five-Letter Word)** Write an application that reads a five-letter word from the user and produces every possible three-letter string that can be derived from the letters of that word. For example, the three-letter words produced from the word “bathe” include “ate,” “bat,” “bet,” “tab,” “hat,” “the” and “tea.”

### Special Section: Advanced String-Manipulation Exercises

The preceding exercises are keyed to the text and designed to test your understanding of fundamental string-manipulation concepts. This section includes a collection of intermediate and advanced string-manipulation exercises. You should find these problems challenging, yet entertaining. The problems vary considerably in difficulty. Some require an hour or two of application writing and implementation. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects.

**14.18 (Text Analysis)** The availability of computers with string-manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. Much attention has been focused on whether William Shakespeare ever lived. Some scholars believe there’s substantial evidence indicating that Christopher Marlowe actually penned the masterpieces attributed to Shakespeare. Researchers have used computers to find similarities in the writings of these two authors. This exercise examines three methods for analyzing texts with a computer.

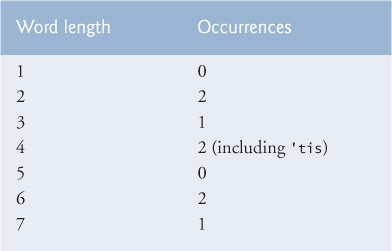
a) Write an application that reads a line of text from the keyboard and prints a table indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0638pro01a)

To be, or not to be: that is the question:

contains one “a,” two “b’s,” no “c’s,” and so on.

b) Write an application that reads a line of text and prints a table indicating the number of one-letter words, two-letter words, three-letter words, and so on, appearing in the text. For example, [Fig. 14.25](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec13_html#ch14fig25)shows the counts for the phrase



**Fig. 14.25** | Word-length counts for the string

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0638pro02a)

Whether 'tis nobler in the mind to suffer

c) Write an application that reads a line of text and prints a table indicating the number of occurrences of each different word in the text. The application should include the words in the table in the same order in which they appear in the text. For example, the lines

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0638pro03a)

To be, or not to be: that is the question:  
Whether 'tis nobler in the mind to suffer

contain the word “to” three times, the word “be” two times, the word “or” once, etc.

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0639pro01a)

"Whether 'tis nobler in the mind to suffer".

**14.19 (Printing Dates in Various Formats)** Dates are printed in several common formats. Two of the more common formats are

04/25/1955 and April 25, 1955

Write an application that reads a date in the first format and prints it in the second format.

**14.20 (Check Protection)** Computers are frequently employed in check-writing systems, such as payroll and accounts payable applications. There are many strange stories about weekly paychecks being printed (by mistake) for amounts in excess of $1 million. Incorrect amounts are printed by computerized check-writing systems because of human error or machine failure. Systems designers build controls into their systems to prevent such erroneous checks from being issued.

Another serious problem is the intentional alteration of a check amount by someone who plans to cash a check fraudulently. To prevent a dollar amount from being altered, some computerized check-writing systems employ a technique called check protection. Checks designed for imprinting by computer contain a fixed number of spaces in which the computer may print an amount. Suppose a paycheck contains eight blank spaces in which the computer is supposed to print the amount of a weekly paycheck. If the amount is large, then all eight of the spaces will be filled. For example,

1,230.60 *(check amount)*  
--------  
12345678 *(position numbers)*

On the other hand, if the amount is less than $1000, then several of the spaces would ordinarily be left blank. For example,

99.87  
-------  
12345678

contains three blank spaces. If a check is printed with blank spaces, it’s easier for someone to alter the amount. To prevent alteration, many check-writing systems insertleading asterisks to protect the amount as follows:

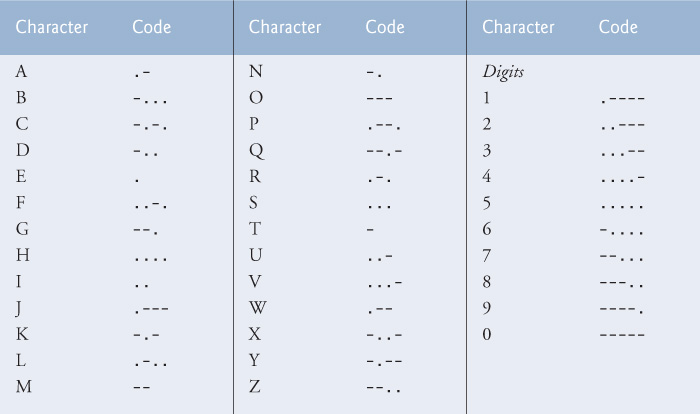
\*\*\*99.87  
-------  
12345678

Write an application that inputs a dollar amount to be printed on a check, then prints the amount in check-protected format with leading asterisks if necessary. Assume that nine spaces are available for printing the amount.

**14.21 (Writing the Word Equivalent of a Check Amount) C**ontinuing the discussion in [Exercise 14.20](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec13_html#ch14que20), we reiterate the importance of designing check-writing systems to prevent alteration of check amounts. One common security method requires that the amount be written in numbers and spelled out in words as well. Even if someone is able to alter the numerical amount of the check, it’s extremely difficult to change the amount in words. Write an application that inputs a numeric check amount that’s less than $1000 and writes the word equivalent of the amount. For example, the amount 112.43 should be written as

ONE hundred TWELVE and 43/100

**14.22 (Morse Code)** Perhaps the most famous of all coding schemes is the Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. The Morse code assigns a series of dots and dashes to each letter of the alphabet, each digit, and a few special characters (e.g., period, comma, colon, semicolon). In sound-oriented systems, the dot represents a short sound and the dash a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems. Separation between words is indicated by a space or, simply, the absence of a dot or dash. In a sound-oriented system, a space is indicated by a short time during which no sound is transmitted. The international version of the Morse code appears in [Fig. 14.26](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec13_html#ch14fig26).



**Fig. 14.26** | Letters and digits as expressed in international Morse code.

Write an application that reads an English-language phrase and encodes it into Morse code. Also write an application that reads a phrase in Morse code and converts it into the English-language equivalent. Use one blank between each Morse-coded letter and three blanks between each Morse-coded word.

**14.23 (Metric Conversions)** Write an application that will assist the user with metric conversions. Your application should allow the user to specify the names of the units as strings (i.e., centimeters, liters, grams, and so on, for the metric system and inches, quarts, pounds, and so on, for the English system) and should respond to simple questions, such as

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0640pro01a)

"How many inches are in 2 meters?"  
"How many liters are in 10 quarts?"

Your application should recognize invalid conversions. For example, the question

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0641pro01a)

"How many feet are in 5 kilograms?"

is not meaningful because "feet" is a unit of length, whereas "kilograms" is a unit of mass.

### Special Section: Challenging String-Manipulation Projects

**14.24****(Project: A Spelling Checker)** Many popular word-processing software packages have built-in spell checkers. In this project, you’re asked to develop your own spell-checker utility. We make suggestions to help get you started. You should then consider adding more capabilities. Use a computerized dictionary (if you have access to one) as a source of words.

Why do we type so many words with incorrect spellings? In some cases, it’s because we simply do not know the correct spelling, so we make a best guess. In some cases, it’s because we transpose two letters (e.g., “defualt” instead of “default”). Sometimes we double-type a letter accidentally (e.g., “hanndy” instead of “handy”). Sometimes we type a nearby key instead of the one we intended (e.g., “biryhday” instead of “birthday”), and so on.

Design and implement a spell-checker application in Java. Your application should maintain an array wordList of strings. Enable the user to enter these strings. [Note: In[Chapter 15](http://proquest.safaribooksonline.com/9780133813036/ch15_html#ch15), we introduce file processing. With this capability, you can obtain the words for the spell checker from a computerized dictionary stored in a file.]

Your application should ask a user to enter a word. The application should then look up that word in the wordListarray. If the word is in the array, your application should print "Word is spelled correctly." If the word is not in the array, your application should print "Word is not spelled correctly." Then your application should try to locate other words in wordList that might be the word the user intended to type. For example, you can try all possible single transpositions of adjacent letters to discover that the word “default” is a direct match to a word in wordList. Of course, this implies that your application will check all other single transpositions, such as “edfault,” “dfeault,” “deafult,” “defalut” and “defautl.” When you find a new word that matches one in wordList, print it in a message, such as

Did you mean "default"?

Implement other tests, such as replacing each double letter with a single letter, and any other tests you can develop to improve the value of your spell checker.

**14.25 (Project: A Crossword Puzzle Generator)** Most people have worked a crossword puzzle, but few have ever attempted to generate one. Generating a crossword puzzle is suggested here as a string-manipulation project requiring substantial sophistication and effort.

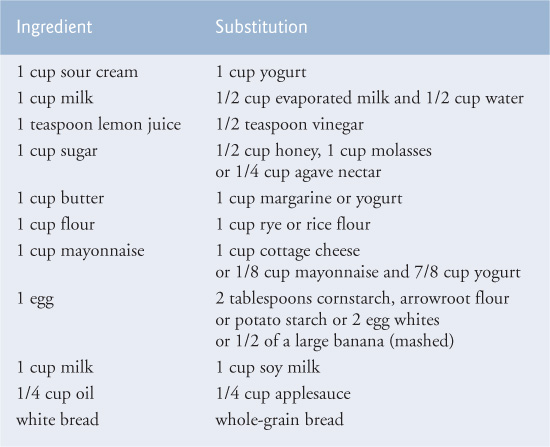
There are many issues the programmer must resolve to get even the simplest crossword-puzzle-generator application working. For example, how do you represent the grid of a crossword puzzle inside the computer? Should you use a series of strings or two-dimensional arrays?

The programmer needs a source of words (i.e., a computerized dictionary) that can be directly referenced by the application. In what form should these words be stored to facilitate the complex manipulations required by the application?

If you’re really ambitious, you’ll want to generate the clues portion of the puzzle, in which the brief hints for each across word and each down word are printed. Merely printing a version of the blank puzzle itself is not a simple problem.

### Making a Difference

**14.26****(Cooking with Healthier Ingredients)** Obesity in America is increasing at an alarming rate. Check the map from the Centers for Disease Control and Prevention (CDC) at[www.cdc.gov/nccdphp/dnpa/Obesity/trend/maps/index.htm](http://www.cdc.gov/nccdphp/dnpa/Obesity/trend/maps/index.htm), which shows obesity trends in the United States over the last 20 years. As obesity increases, so do occurrences of related problems (e.g., heart disease, high blood pressure, high cholesterol, type 2 diabetes). Write a program that helps users choose healthier ingredients when cooking, and helps those allergic to certain foods (e.g., nuts, gluten) find substitutes. The program should read a recipe from aJTextArea and suggest healthier replacements for some of the ingredients. For simplicity, your program should assume the recipe has no abbreviations for measures such as teaspoons, cups, and tablespoons, and uses numerical digits for quantities (e.g., 1 egg, 2 cups) rather than spelling them out (one egg, two cups). Some common substitutions are shown in [Fig. 14.27](http://proquest.safaribooksonline.com/9780133813036/ch14lev1sec15_html#ch14fig27). Your program should display a warning such as, “Always consult your physician before making significant changes to your diet.”



**Fig. 14.27** | Common food substitutions.

Your program should take into consideration that replacements are not always one-for-one. For example, if a cake recipe calls for three eggs, it might reasonably use six egg whites instead. Conversion data for measurements and substitutes can be obtained at websites such as:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0642pro01a)

chinesefood.about.com/od/recipeconversionfaqs/f/usmetricrecipes.htm  
www.pioneerthinking.com/eggsub.html  
www.gourmetsleuth.com/conversions.htm

Your program should consider the user’s health concerns, such as high cholesterol, high blood pressure, weight loss, gluten allergy, and so on. For high cholesterol, the program should suggest substitutes for eggs and dairy products; if the user wishes to lose weight, low-calorie substitutes for ingredients such as sugar should be suggested.

**14.27 (Spam Scanner)** Spam (or junk e-mail) costs U.S. organizations billions of dollars a year in spam-prevention software, equipment, network resources, bandwidth, and lost productivity. Research online some of the most common spam e-mail messages and words, and check your own junk e-mail folder. Create a list of 30 words and phrases commonly found in spam messages. Write an application in which the user enters an e-mail message in a JTextArea. Then, scan the message for each of the 30 keywords or phrases. For each occurrence of one of these within the message, add a point to the message’s “spam score.” Next, rate the likelihood that the message is spam, based on the number of points it received.

**14.28 (SMS Language)** Short Message Service (SMS) is a communications service that allows sending text messages of 160 or fewer characters between mobile phones. With the proliferation of mobile phone use worldwide, SMS is being used in many developing nations for political purposes (e.g., voicing opinions and opposition), reporting news about natural disasters, and so on. For example, check outcomunica.org/radio2.0/archives/87. Since the length of SMS messages is limited, SMS Language—abbreviations of common words and phrases in mobile text messages, emails, instant messages, etc.—is often used. For example, “in my opinion” is “imo” in SMS Language. Research SMS Language online. Write a GUI application in which the user can enter a message using SMS Language, then click a button to translate it into English (or your own language). Also provide a mechanism to translate text written in English (or your own language) into SMS Language. One potential problem is that one SMS abbreviation could expand into a variety of phrases. For example, IMO (as used above) could also stand for “International Maritime Organization,” “in memory of,” etc.