4.1 Introduction

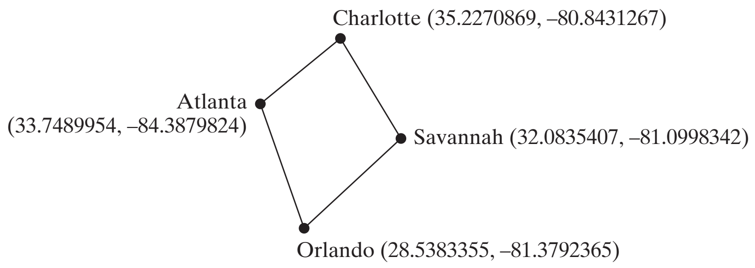
Key Point

The focus of this chapter is to introduce mathematical functions, characters, string objects, and use them to develop programs.

The preceding chapters introduced fundamental programming techniques and taught you how to write simple programs to solve basic problems using selection statements. This chapter introduces methods for performing common mathematical operations. You will learn how to create custom methods in Chapter 6.

problem

Suppose you need to estimate the area enclosed by four cities, given the GPS locations (latitude and longitude) of these cities, as shown in the following diagram. How would you write a program to solve this problem? You will be able to write such a program after completing this chapter.


Charlotte (35.2270869, –80.8431267)
Atlanta
(33.7489954, –84.3879824)
Savannah (32.0835407, –81.0998342)
Orlando (28.5383355, –81.3792365)

Because strings are frequently used in programming, it is beneficial to introduce strings early so that you can begin to use them to develop useful programs. This chapter gives a brief introduction to string objects; you will learn more on objects and strings in Chapters 9 and 10.

4.2 Common Mathematical Functions

Key Point

Java provides many useful methods in the `Math` *class for performing common mathematical functions.*

A method is a group of statements that performs a specific task. You have already used the `pow(a, b)` method to compute a^b in Section 2.9.4, Exponent Operations and the `random()` method for generating a random number in Section 3.7. This section introduces other useful methods in the `Math` class. They can be categorized as *trigonometric methods*, *exponent methods*, and *service methods*. Service methods include the rounding, min, max, absolute, and random methods. In addition to methods, the `Math` class provides two useful `double` constants, `PI` and `E` (the base of natural logarithms). You can use these constants as `Math.PI` and `Math.E` in any program.

4.2.1 Trigonometric Methods

VideoNote
Introduce math functions

The `Math` class contains the following methods as shown in Table 4.1 for performing trigonometric functions:

TABLE 4.1 Trigonometric Methods in the Math Class

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

The parameter for `sin`, `cos`, and `tan` is an angle in radians. The return value for `asin`, `acos`, and `atan` is a degree in radians in the range between $-\pi/2$ and $\pi/2$. One degree is equal to $\pi/180$ in radians, 90 degrees is equal to $\pi/2$ in radians, and 30 degrees is equal to $\pi/6$ in radians.

For example,

```
Math.toDegrees(Math.PI / 2) returns 90.0
Math.toRadians(30) returns 0.5236 (same as π/6)
Math.sin(0) returns 0.0
Math.sin(Math.toRadians(270)) returns -1.0
Math.sin(Math.PI / 6) returns 0.5
Math.sin(Math.PI / 2) returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6) returns 0.866
Math.cos(Math.PI / 2) returns 0
Math.asin(0.5) returns 0.523598333 (same as π/6)
Math.acos(0.5) returns 1.0472 (same as π/3)
Math.atan(1.0) returns 0.785398 (same as π/4)
```

4.2.2 Exponent Methods

There are five methods related to exponents in the `Math` class as shown in Table 4.2.

TABLE 4.2 Exponent Methods in the Math Class

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

For example,

```
Math.exp(1) returns 2.71828
Math.log(Math.E) returns 1.0
Math.log10(10) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(4.5, 2.5) returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 4.24
```

4.2.3 The Rounding Methods

The `Math` class contains five rounding methods as shown in Table 4.3.

TABLE 4.3 Rounding Methods in the Math Class

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

For example,

```
Math.ceil(2.1) returns 4.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -4.0
Math.rint(2.1) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(4.5) returns 4.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 // Returns int
Math.round(2.0) returns 2 // Returns long
Math.round(-2.0f) returns -2 // Returns int
Math.round(-2.6) returns -3 // Returns long
Math.round(-2.4) returns -2 // Returns long
```

4.2.4 The **`min`**, **`max`**, and **`abs`** Methods

The `min` and `max` methods return the minimum and maximum numbers of two numbers (`int`, `long`, `float`, or `double`). For example, `max(4.4, 5.0)` returns `5.0`, and `min(3, 2)` returns `2`.

The `abs` method returns the absolute value of the number (`int`, `long`, `float`, or `double`). For example,

```
Math.max(2, 3) returns 3
Math.max(2.5, 3) returns 4.0
Math.min(2.5, 4.6) returns 2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1
```

4.2.5 The **`random`** Method

You have used the `random()` method in the preceding chapter. This method generates a random `double` value greater than or equal to 0.0 and less than 1.0 (`0 <= Math.random() < 1.0`). You can use it to write a simple expression to generate random numbers in any range. For example,

`(int)(Math.random() * 10)` $\longrightarrow$ Returns a random integer between `0` and `9`.

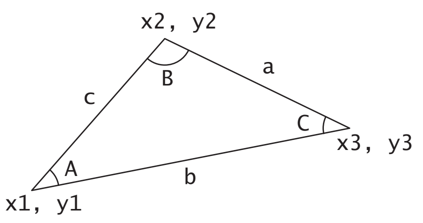
`50 + (int)(Math.random() * 50)` $\longrightarrow$ Returns a random integer between `50` and `99`.

In general,

`a + Math.random() * b` $\longrightarrow$ Returns a random number between `a` and `a + b`, excluding `a + b`.

4.2.6 Case Study: Computing Angles of a Triangle

You can use the math methods to solve many computational problems. Given the three sides of a triangle, for example, you can compute the angles by using the following formula:



```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

Don't be intimidated by the mathematic formula. As we discussed early in Listing 2.9, ComuteLoan.java, you don't have to know how the mathematical formula is derived in order to write a program for computing the loan payments. Here in this example, given the length of three sides, you can use this formula to write a program to compute the angles without having to know how the formula is derived. In order to compute the lengths of the sides, we need to know the coordinates of three corner points and compute the distances between the points.

Listing 4.1 is an example of a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the three angles.

LISTING 4.1 ComputeAngles.java

```
import java.util.Scanner;

public class ComputeAngles {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter three points
    System.out.print("Enter three points: ");
    double x1 = input.nextDouble();
    double y1 = input.nextDouble();
    double x2 = input.nextDouble();
    double y2 = input.nextDouble();
    double x3 = input.nextDouble();
    double y3 = input.nextDouble();

    // Compute three sides
    double a = Math.sqrt((x2 - x3) * (x2 - x3)
        + (y2 - y3) * (y2 - y3));
    double b = Math.sqrt((x1 - x3) * (x1 - x3)
        + (y1 - y3) * (y1 - y3));
    double c = Math.sqrt((x1 - x2) * (x1 - x2)
        + (y1 - y2) * (y1 - y2));

    // Compute three angles
    double A = Math.toDegrees(Math.acos((a * a - b * b - c * c)
        / (-2 * b * c)));
    double B = Math.toDegrees(Math.acos((b * b - a * a - c * c)
        / (-2 * a * c)));
    double C = Math.toDegrees(Math.acos((c * c - b * b - a * a)
        / (-2 * a * b)));

    // Display results
    System.out.println("The three angles are " +
        Math.round(A * 100) / 100.0 + " " +
```

enter three points

compute sides

display result

```
35          Math.round(B * 100) / 100.0 + " " +
36          Math.round(C * 100) / 100.0);
37    }
38  }
```

```
Enter three points: 1 1 6.5 1 6.5 2.5 ↵Enter
The three angles are 15.26 90.0 74.74
```

The program prompts the user to enter three points (line 8). This prompting message is not clear. You should give the user explicit instructions on how to enter these points as follows:

```
System.out.print("Enter the coordinates of three points separated "
    + "by spaces like x1 y1 x2 y2 x3 y3: ");
```

Note that the distance between two points `(x1, y1)` and `(x2, y2)` can be computed using the formula $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. The program computes the distances between two points (lines 17–22), and applies the formula to compute the angles (lines 25–30). The angles are rounded to display up to two digits after the decimal point (lines 34–36).

The `Math` class is used in the program, but not imported, because it is in the `java.lang` package. All the classes in the `java.lang` package are *implicitly* imported in a Java program.

Check Point

4.1 Evaluate the following method calls:

(a) `Math.sqrt(4)`
(b) `Math.sin(2 * Math.PI)`
(c) `Math.cos(2 * Math.PI)`
(d) `Math.pow(2, 2)`
(e) `Math.log(Math.E)`
(f) `Math.exp(1)`
(g) `Math.max(2, Math.min(3, 4))`
(h) `Math.rint(-2.5)`
(i) `Math.ceil(-2.5)`
(j) `Math.floor(-2.5)`
(k) `Math.round(-2.5f)`
(l) `Math.round(-2.5)`
(m) `Math.rint(2.5)`
(n) `Math.ceil(2.5)`
(o) `Math.floor(2.5)`
(p) `Math.round(2.5f)`
(q) `Math.round(2.5)`
(r) `Math.round(Math.abs(-2.5))`

4.2 True or false? The argument for trigonometric methods is an angle in radians.

4.3 Write a statement that converts `47` degrees to radians and assigns the result to a variable.

4.4 Write a statement that converts `π / 7` to an angle in degrees and assigns the result to a variable.

4.5 Write an expression that obtains a random integer between `34` and `55`. Write an expression that obtains a random integer between `0` and `999`. Write an expression that obtains a random number between `5.5` and `55.5`.

4.6 Why does the `Math` class not need to be imported?

4.7 What is `Math.log(Math.exp(5.5))`? What is `Math.exp(Math.log(5.5))`? What is `Math.asin(Math.sin(Math.PI / 6))`? What is `Math.sin(Math.asin(Math.PI / 6))`?

4.3 Character Data Type and Operations

A character data type represents a single character.

Key Point

In addition to processing numeric values, you can process characters in Java. The character data type, **char**, is used to represent a single character. A character literal is enclosed in single quotation marks. Consider the following code:

char type

```
char letter = 'A';
char numChar = '4';
```

The first statement assigns character **A** to the **char** variable **letter**. The second statement assigns digit character **4** to the **char** variable **numChar**.

Caution

A string literal must be enclosed in quotation marks (**" "**). A character literal is a single character enclosed in single quotation marks (**' '**). Therefore, **"A"** is a string, but **'A'** is a character.

char literal

4.3.1 Unicode and ASCII code

Computers use binary numbers internally. A character is stored in a computer as a sequence of 0s and 1s. Mapping a character to its binary representation is called *encoding*. There are different ways to encode a character. How characters are encoded is defined by an *encoding scheme*.

encoding

Java supports *Unicode*, an encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode was originally designed as a 16-bit character encoding. The primitive data type **char** was intended to take advantage of this design by providing a simple data type that could hold any character. However, it turned out that the 65,536 characters possible in a 16-bit encoding are not sufficient to represent all the characters in the world. The Unicode standard therefore has been extended to allow up to 1,112,064 characters. Those characters that go beyond the original 16-bit limit are called *supplementary characters*. Java supports the supplementary characters. The processing and representing of supplementary characters are beyond the scope of this book. For simplicity, this book considers only the original 16-bit Unicode characters. These characters can be stored in a **char** type variable.

Unicode

original Unicode

supplementary Unicode

A 16-bit Unicode takes two bytes, preceded by **\u**, expressed in four hexadecimal digits that run from **\u0000** to **\uFFFF**. Hexadecimal numbers are introduced in Appendix F, Number Systems. For example, the English word **welcome** is translated into Chinese using two characters, 欢迎. The Unicodes of these two characters are **\u6B22\u8FCE**. The Unicodes for the Greek letters α β γ are **\u03b1 \u03b2 \u03b4**.

Most computers use *ASCII* (*American Standard Code for Information Interchange*), an 8-bit encoding scheme for representing all uppercase and lowercase letters, digits, punctuation marks, and control characters. Unicode includes ASCII code, with **\u0000** to **\u007F** corresponding to the 128 ASCII characters. Table 4.4 shows the ASCII code for some commonly used characters. Appendix B, 'The ASCII Character Set,' gives a complete list of ASCII characters and their decimal and hexadecimal codes.

TABLE 4.4 ASCII Code for Commonly Used Characters

Characters	*Code Value in Decimal*	*Unicode Value*
'0' to **'9'**	48 to 57	\u0030 to \u0039
'A' to **'Z'**	65 to 90	\u0041 to \u005A
'a' to **'z'**	97 to 122	\u0061 to \u007A

ASCII

You can use ASCII characters such as `'X'`, `'1'`, and `'$'` in a Java program as well as Unicodes. Thus, for example, the following statements are equivalent:

```
char letter = 'A';
char letter = '\u0041'; // Character A's Unicode is 0041
```

Both statements assign character `A` to the `char` variable `letter`.

`char` increment and decrement

Note

The increment and decrement operators can also be used on `char` variables to get the next or preceding Unicode character. For example, the following statements display character `b`.

```
char ch = 'a';
System.out.println(++ch);
```

4.3.2 Escape Sequences for Special Characters

Suppose you want to print a message with quotation marks in the output. Can you write a statement like this?

```
System.out.println("He said "Java is fun"");
```

No, this statement has a compile error. The compiler thinks the second quotation character is the end of the string and does not know what to do with the rest of characters.

escape sequence

To overcome this problem, Java uses a special notation to represent special characters, as shown in Table 4.5. This special notation, called an *escape sequence*, consists of a backslash (`\`) followed by a character or a combination of digits. For example, `\t` is an escape sequence for the Tab character and an escape sequence such as `\u03b1` is used to represent a Unicode. The symbols in an escape sequence are interpreted as a whole rather than individually. An escape sequence is considered as a single character.

So, now you can print the quoted message using the following statement:

```
System.out.println("He said \"Java is fun\"");
```

The output is

```
He said "Java is fun"
```

Note that the symbols `\` and `"` together represent one character.

TABLE 4.5 Escape Sequences

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
`\b`	Backspace	`\u0008`	8
`\t`	Tab	`\u0009`	9
`\n`	Linefeed	`\u000A`	10
`\f`	Formfeed	`\u000C`	12
`\r`	Carriage Return	`\u000D`	13
`\\`	Backslash	`\u005C`	92
`\"`	Double Quote	`\u0022`	34

escape character

The backslash `\` is called an *escape character*. It is a special character. To display this character, you have to use an escape sequence `\\`. For example, the following code

```
System.out.println("\\t is a tab character");
```

displays

```
\t is a tab character
```

4.3.3 Casting between **char** and Numeric Types

A `char` can be cast into any numeric type, and vice versa. When an integer is cast into a `char`, only its lower 16 bits of data are used; the other part is ignored. For example:

```
char ch = (char)0XAB0041; // The lower 16 bits hex code 0041 is
                          // assigned to ch
System.out.println(ch);   // ch is character A
```

When a floating-point value is cast into a `char`, the floating-point value is first cast into an `int`, which is then cast into a `char`.

```
char ch = (char)65.25;    // Decimal 65 is assigned to ch
System.out.println(ch);   // ch is character A
```

When a `char` is cast into a numeric type, the character's Unicode is cast into the specified numeric type.

```
int i = (int)'A'; // The Unicode of character A is assigned to i
System.out.println(i);  // i is 65
```

Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used. For example, since the Unicode of `'a'` is `97`, which is within the range of a byte, these implicit castings are fine:

```
byte b = 'a';
int i = 'a';
```

But the following casting is incorrect, because the Unicode `\uFFF4` cannot fit into a byte:

```
byte b = '\uFFF4';
```

To force this assignment, use explicit casting, as follows:

```
byte b = (byte)'\uFFF4';
```

Any positive integer between `0` and `FFFF` in hexadecimal can be cast into a character implicitly. Any number not in this range must be cast into a `char` explicitly.

numeric operators on characters

All numeric operators can be applied to `char` operands. A `char` operand is automatically cast into a number if the other operand is a number or a character. If the other operand is a string, the character is concatenated with the string. For example, the following statements

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
System.out.println("i is " + i); // i is 101
int j = 2 + 'a'; // (int)'a' is 97
System.out.println("j is " + j); // j is 99
System.out.println(j + " is the Unicode for character "
  + (char)j); // 99 is the Unicode for character c
System.out.println("Chapter " + '2');
```

display

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

4.3.4 Comparing and Testing Characters

Two characters can be compared using the relational operators just like comparing two numbers. This is done by comparing the Unicodes of the two characters. For example,

`'a' < 'b'` is true because the Unicode for `'a'` (97) is less than the Unicode for `'b'` (98).

`'a' < 'A'` is false because the Unicode for `'a'` (97) is greater than the Unicode for `'A'` (65).

`'1' < '8'` is true because the Unicode for `'1'` (49) is less than the Unicode for `'8'` (56).

Often in the program, you need to test whether a character is a number, a letter, an uppercase letter, or a lowercase letter. As shown in Appendix B, the ASCII character set, that the Unicodes for lowercase letters are consecutive integers starting from the Unicode for `'a'`, then for `'b'`, `'c'`, . . ., and `'z'`. The same is true for the uppercase letters and for numeric characters. This property can be used to write the code to test characters. For example, the following code tests whether a character `ch` is an uppercase letter, a lowercase letter, or a digital character.

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

For convenience, Java provides the following methods in the `Character` class for testing characters as shown in Table 4.6.

TABLE 4.6 Methods in the Character Class

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

For example,

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
System.out.println("isLowerCase('a') is "
  + Character.isLowerCase('a'));
System.out.println("isUpperCase('a') is "
  + Character.isUpperCase('a'));
System.out.println("toLowerCase('T') is "
  + Character.toLowerCase('T'));
System.out.println("toUpperCase('q') is "
  + Character.toUpperCase('q'));
```

displays

```
isDigit('a') is false
isLetter('a') is true
```

```
isLowerCase('a') is true
isUpperCase('a') is false
toLowerCase('T') is t
toUpperCase('q') is Q
```

Check Point

4.8 Use print statements to find out the ASCII code for `'1'`, `'A'`, `'B'`, `'a'`, and `'b'`. Use print statements to find out the character for the decimal codes `40`, `59`, `79`, `85`, and `90`. Use print statements to find out the character for the hexadecimal code `40`, `5A`, `71`, `72`, and `7A`.

4.9 Which of the following are correct literals for characters?

```
'1', '\u345dE', '\u3fFa', '\b', '\t'
```

4.10 How do you display the characters `\` and `"`?

4.11 Evaluate the following:

```
int i = '1';
int j = '1' + '2' * ('4' - '3') + 'b' / 'a';
int k = 'a';
char c = 90;
```

4.12 Can the following conversions involving casting be allowed? If so, find the converted result.

```
char c = 'A';
int i = (int)c;

float f = 1000.34f;
int i = (int)f;

double d = 1000.34;
int i = (int)d;

int i = 97;
char c = (char)i;
```

4.13 Show the output of the following program:

```
public class Test {
  public static void main(String[] args) {
    char x = 'a';
    char y = 'c';
    System.out.println(++x);
    System.out.println(y++);
    System.out.println(x - y);
  }
}
```

4.14 Write the code that generates a random lowercase letter.

4.15 Show the output of the following statements:

```
System.out.println('a' < 'b');
System.out.println('a' <= 'A');
System.out.println('a' > 'b');
System.out.println('a' >= 'A');
System.out.println('a' == 'a');
System.out.println('a' != 'b');
```

4.4 The String Type

Key Point

A string is a sequence of characters.

The `char` type represents only one character. To represent a string of characters, use the data type called `String`. For example, the following code declares `message` to be a string with the value `"Welcome to Java"`.

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

VideoNote
Introduce strings and objects

`String` is a predefined class in the Java library, just like the classes `System` and `Scanner`. The `String` type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. The variable declared by a reference type is known as a reference variable that references an object. Here, `message` is a reference variable that references a string object with contents `Welcome to Java`.

Reference data types will be discussed in detail in Chapter 9, Objects and Classes. For the time being, you need to know only how to declare a `String` variable, how to assign a string to the variable, and how to use the methods in the `String` class. More details on using strings will be covered in Chapter 10.

Table 4.7 lists the `String` methods for obtaining string length, for accessing characters in the string, for concatenating strings, for converting a string to upper or lowercases, and for trimming a string.

TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

instance method
static method

Strings are objects in Java. The methods in Table 4.7 can only be invoked from a specific string instance. For this reason, these methods are called *instance methods*. A non-instance method is called a *static method*. A static method can be invoked without using an object. All the methods defined in the `Math` class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is `referenceVariable.methodName(arguments)`. A method may have many arguments or no arguments. For example, the `charAt(index)` method has one argument, but the `length()` method has no arguments. Recall that the syntax to invoke a static method is `ClassName.methodName(arguments)`. For example, the `pow` method in the `Math` class can be invoked using `Math.pow(2, 2.5)`.

4.4.1 Getting String Length

You can use the `length()` method to return the number of characters in a string. For example, the following code

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "
  + message.length());
```

displays

```
The length of Welcome to Java is 15
```

Note

When you use a string, you often know its literal value. For convenience, Java allows you to use the string literal to refer directly to strings without creating new variables. Thus, `"Welcome to Java".length()` is correct and returns `15`. Note that `""` denotes an *empty string* and `"".length()` is `0`.

string literal

empty string

4.4.2 Getting Characters from a String

charAt(index)

The `s.charAt(index)` method can be used to retrieve a specific character in a string `s`, where the index is between `0` and `s.length()-1`. For example, `message.charAt(0)` returns the character `W`, as shown in Figure 4.1. Note that the index for the first character in the string is `0`.

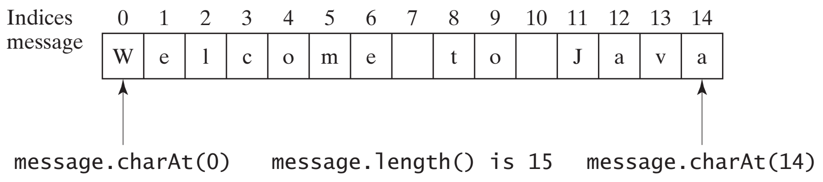


FIGURE 4.1 The characters in a `String` object can be accessed using its index.

Caution

Attempting to access characters in a string `s` out of bounds is a common programming error. To avoid it, make sure that you do not use an index beyond `s.length() – 1`. For example, `s.charAt(s.length())` would cause a `StringIndexOutOfBoundsException`.

string index range

4.4.3 Concatenating Strings

You can use the `concat` method to concatenate two strings. The statement shown below, for example, concatenates strings `s1` and `s2` into `s3`:

```
String s3 = s1.concat(s2);
```

s1.concat(s2)

Because string concatenation is heavily used in programming, Java provides a convenient way to accomplish it. You can use the plus (`+`) operator to concatenate two strings, so the previous statement is equivalent to

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

s1 + s2

The following code combines the strings `message`, `" and "`, and `"HTML"` into one string:

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

Recall that the `+` operator can also concatenate a number with a string. In this case, the number is converted into a string and then concatenated. Note that at least one of the operands must be a string in order for concatenation to take place. If one of the operands is a nonstring

concatenate strings and numbers

(e.g., a number), the nonstring value is converted into a string and concatenated with the other string. Here are some examples:

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

If neither of the operands is a string, the plus sign (`+`) is the addition operator that adds two numbers.

The augmented `+=` operator can also be used for string concatenation. For example, the following code appends the string `"and Java is fun"` with the string `"Welcome to Java"` in `message`.

```
message += " and Java is fun";
```

So the new `message` is `"Welcome to Java and Java is fun"`.

If `i = 1` and `j = 2`, what is the output of the following statement?

```
System.out.println("i + j is " + i + j);
```

The output is `"i + j is 12"` because `"i + j is "` is concatenated with the value of `i` first. To force `i + j` to be executed first, enclose `i + j` in the parentheses, as follows:

```
System.out.println("i + j is " + (i + j));
```

4.4.4 Converting Strings

The `toLowerCase()` method returns a new string with all lowercase letters and the `toUpperCase()` method returns a new string with all uppercase letters. For example,

toLowerCase()
toUpperCase()

`"Welcome".toLowerCase()` returns a new string `welcome`.
`"Welcome".toUpperCase()` returns a new string `WELCOME`.

whitespace character

The `trim()` method returns a new string by eliminating whitespace characters from both ends of the string. The characters `' '`, `\t`, `\f`, `\r`, or `\n` are known as *whitespace characters*. For example,

trim()

`"\t Good Night \n".trim()` returns a new string `Good Night`.

4.4.5 Reading a String from the Console

read strings

To read a string from the console, invoke the `next()` method on a `Scanner` object. For example, the following code reads three strings from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

```
Enter three words separated by spaces: Welcome to Java ↵Enter
s1 is Welcome
s2 is to
s3 is Java
```

The `next()` method reads a string that ends with a whitespace character. You can use the `nextLine()` method to read an entire line of text. The `nextLine()` method reads a string that ends with the *Enter* key pressed. For example, the following statements read a line of text.

whitespace character

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

```
Enter a line: Welcome to Java ↵Enter
The line entered is Welcome to Java
```

Important Caution

To *avoid input errors*, do not use `nextLine()` after `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `next()`. The reasons will be explained in Section 12.11.4, 'How Does `Scanner` Work?'

avoid input errors

4.4.6 Reading a Character from the Console

To read a character from the console, use the `nextLine()` method to read a string and then invoke the `charAt(0)` method on the string to return a character. For example, the following code reads a character from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

4.4.7 Comparing Strings

The `String` class contains the methods as shown in Table 4.8 for comparing two strings.

TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if s1 is a substring in this string.

How do you compare the contents of two strings? You might attempt to use the == operator, as follows:

==

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

However, the == operator checks only whether string1 and string2 refer to the same object; it does not tell you whether they have the same contents. Therefore, you cannot use the == operator to find out whether two string variables have the same contents. Instead, you should use the equals method. The following code, for instance, can be used to compare two strings:

string1.equals(string2)

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

For example, the following statements display true and then false.

```
String s1 = "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
```

The compareTo method can also be used to compare two strings. For example, consider the following code:

s1.compareTo(s2)

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

The method returns the value 0 if s1 is equal to s2, a value less than 0 if s1 is lexicographically (i.e., in terms of Unicode ordering) less than s2, and a value greater than 0 if s1 is lexicographically greater than s2.

The actual value returned from the compareTo method depends on the offset of the first two distinct characters in s1 and s2 from left to right. For example, suppose s1 is abc and s2 is abg, and s1.compareTo(s2) returns -4. The first two characters (a vs. a) from s1 and s2 are compared. Because they are equal, the second two characters (b vs. b) are compared. Because they are also equal, the third two characters (c vs. g) are compared. Since the character c is 4 less than g, the comparison returns -4.

Caution

Syntax errors will occur if you compare strings by using relational operators >, >=, <, or <=. Instead, you have to use s1.compareTo(s2).

Note

The equals method returns true if two strings are equal and false if they are not. The compareTo method returns 0, a positive integer, or a negative integer, depending on whether one string is equal to, greater than, or less than the other string.

The String class also provides the equalsIgnoreCase and compareToIgnoreCase methods for comparing strings. The equalsIgnoreCase and compareToIgnoreCase methods ignore the case of the letters when comparing two strings. You can also use str.startsWith(prefix) to check whether string str starts with a specified prefix, str.endsWith(suffix) to check whether string str ends with a specified suffix, and str.contains(s1) to check whether string str contains string s1. For example,

```
"Welcome to Java".startsWith("We") returns true.
"Welcome to Java".startsWith("we") returns false.
"Welcome to Java".endsWith("va") returns true.
```

```
"Welcome to Java".endsWith("v") returns false.
"Welcome to Java".contains("to") returns true.
"Welcome to Java".contains("To") returns false.
```

Listing 4.2 gives a program that prompts the user to enter two cities and displays them in alphabetical order.

LISTING 4.2 OrderTwoCities.java

```
 1  import java.util.Scanner;
 2
 3  public class OrderTwoCities {
 4    public static void main(String[] args) {
 5      Scanner input = new Scanner(System.in);
 6
 7      // Prompt the user to enter two cities
 8      System.out.print("Enter the first city: ");
 9      String city1 = input.nextLine();
10      System.out.print("Enter the second city: ");
11      String city2 = input.nextLine();
12
13      if (city1.compareTo(city2) < 0)
14        System.out.println("The cities in alphabetical order are " +
15           city1 + " " + city2);
16      else
17        System.out.println("The cities in alphabetical order are " +
18            city2 + " " + city1);
19    }
20  }
```

input city1

input city2

compare two cities

```
Enter the first city: New York ↵Enter
Enter the second city: Boston ↵Enter
The cities in alphabetical order are Boston New York
```

The program reads two strings for two cities (lines 9, 11). If **input.nextLine()** is replaced by **input.next()** (line 9), you cannot enter a string with spaces for **city1**. Since a city name may contain multiple words separated by spaces, the program uses the **nextLine** method to read a string (lines 9, 11). Invoking **city1.compareTo(city2)** compares two strings **city1** with **city2** (line 13). A negative return value indicates that **city1** is less than **city2**.

4.4.8 Obtaining Substrings

You can obtain a single character from a string using the **charAt** method. You can also obtain a substring from a string using the **substring** method in the **String** class, as shown in Table 4.9.

For example,

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

The string **message** now becomes **Welcome to HTML**.

TABLE 4.9 The **String** class contains the methods for obtaining substrings.

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex – 1`, as shown in Figure 4.2. Note that the character at `endIndex` is not part of the substring.

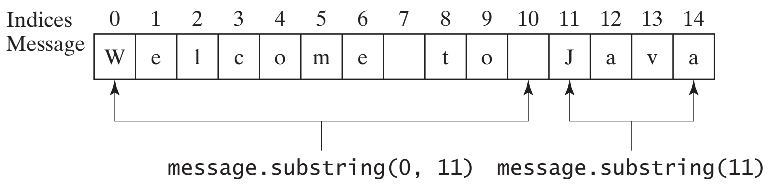


FIGURE 4.2 The `substring` method obtains a substring from a string.

beginIndex <= endIndex

Note

If `beginIndex` is `endIndex`, `substring(beginIndex, endIndex)` returns an empty string with length `0`. If `beginIndex` > `endIndex`, it would be a runtime error.

4.4.9 Finding a Character or a Substring in a String

The `String` class provides several versions of `indexOf` and `lastIndexOf` methods to find a character or a substring in a string, as shown in Table 4.10.

TABLE 4.10 The **String** class contains the methods for finding substrings.

Method	*Description*
`index(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

For example,

`indexOf`

`"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`.

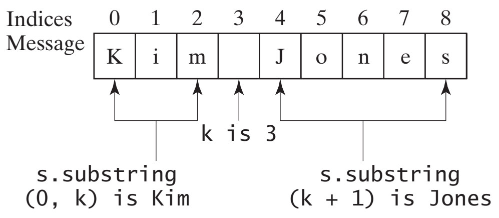
`lastIndexOf`

`"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`.

Suppose a string `s` contains the first name and last name separated by a space. You can use the following code to extract the first name and last name from the string:

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

For example, if s is Kim Jones, the following diagram illustrates how the first name and last name are extracted.



4.4.10 Conversion between Strings and Numbers

You can convert a numeric string into a number. To convert a string into an int value, use the Integer.parseInt method, as follows:

Integer.parseInt method

```
int intValue = Integer.parseInt(intString);
```

where intString is a numeric string such as "123".

To convert a string into a double value, use the Double.parseDouble method, as follows:

Double.parseDouble method

```
double doubleValue = Double.parseDouble(doubleString);
```

where doubleString is a numeric string such as "123.45".

If the string is not a numeric string, the conversion would cause a runtime error. The Integer and Double classes are both included in the java.lang package, and thus they are automatically imported.

You can convert a number into a string, simply use the string concatenating operator as follows:

```
String s = number + "";
```

number to string

Check Point

4.16 Suppose that s1, s2, and s3 are three strings, given as follows:

```
String s1 = "Welcome to Java";
String s2 = "Programming is fun";
String s3 = "Welcome to Java";
```

What are the results of the following expressions?

(a) s1 == s2
(b) s2 == s3
(c) s1.equals(s2)
(d) s1.equals(s3)
(e) s1.compareTo(s2)
(f) s2.compareTo(s3)
(g) s2.compareTo(s2)
(h) s1.charAt(0)
(i) s1.indexOf('j')
(j) s1.indexOf("to")
(k) s1.lastIndexOf('a')
(l) s1.lastIndexOf("o", 15)
(m) s1.length()
(n) s1.substring(5)
(o) s1.substring(5, 11)
(p) s1.startsWith("Wel")
(q) s1.endsWith("Java")
(r) s1.toLowerCase()
(s) s1.toUpperCase()
(t) s1.concat(s2)
(u) s1.contains(s2)
(v) "\t Wel \t".trim()

4.17 Suppose that s1 and s2 are two strings. Which of the following statements or expressions are incorrect?

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

4.18 Show the output of the following statements (write a program to verify your results):

```
System.out.println("1" + 1);
System.out.println('1' + 1);
System.out.println("1" + 1 + 1);
System.out.println("1" + (1 + 1));
System.out.println('1' + 1 + 1);
```

4.19 Evaluate the following expressions (write a program to verify your results):

```
1 + "Welcome " + 1 + 1
1 + "Welcome " + (1 + 1)
1 + "Welcome " + ('\u0001' + 1)
1 + "Welcome " + 'a' + 1
```

4.20 Let s1 be " Welcome " and s2 be " welcome ". Write the code for the following statements:

(a) Check whether s1 is equal to s2 and assign the result to a Boolean variable isEqual.

(b) Check whether s1 is equal to s2, ignoring case, and assign the result to a Boolean variable isEqual.

(c) Compare s1 with s2 and assign the result to an int variable x.

(d) Compare s1 with s2, ignoring case, and assign the result to an int variable x.

(e) Check whether s1 has the prefix AAA and assign the result to a Boolean variable b.

(f) Check whether s1 has the suffix AAA and assign the result to a Boolean variable b.

(g) Assign the length of s1 to an int variable x.

(h) Assign the first character of s1 to a char variable x.

(i) Create a new string s3 that combines s1 with s2.

(j) Create a substring of s1 starting from index 1.

(k) Create a substring of s1 from index 1 to index 4.

(l) Create a new string s3 that converts s1 to lowercase.

(m) Create a new string s3 that converts s1 to uppercase.

(n) Create a new string s3 that trims whitespace characters on both ends of s1.

(o) Assign the index of the first occurrence of the character **e** in **s1** to an **int** variable **x**.

(p) Assign the index of the last occurrence of the string **abc** in **s1** to an **int** variable **x**.

4.5 Case Studies

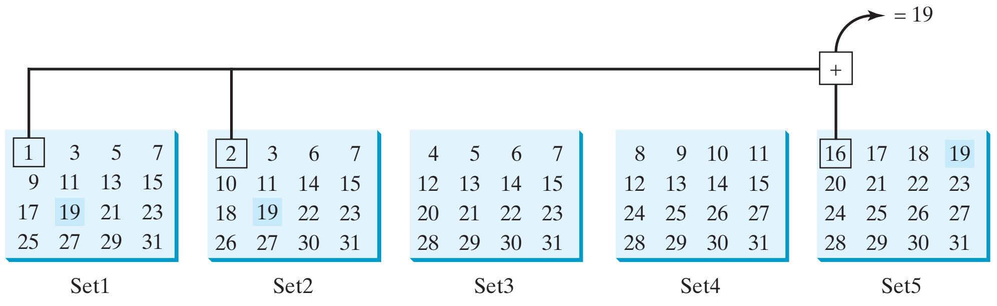
Strings are fundamental in programming. The ability to write programs using strings is essential in learning Java programming.

Key Point

You will frequently use strings to write useful programs. This section presents three examples of solving problems using strings.

4.5.1 Case Study: Guessing Birthdays

You can find out the date of the month when your friend was born by asking five questions. Each question asks whether the day is in one of the five sets of numbers.



The birthday is the sum of the first numbers in the sets where the day appears. For example, if the birthday is **19**, it appears in Set1, Set2, and Set5. The first numbers in these three sets are **1**, **2**, and **16**. Their sum is **19**.

Listing 4.3 gives a program that prompts the user to answer whether the day is in Set1 (lines 41–44), in Set2 (lines 50–53), in Set3 (lines 59–62), in Set4 (lines 68–71), and in Set5 (lines 77–80). If the number is in the set, the program adds the first number in the set to **day** (lines 47, 56, 65, 74, 83).

LISTING 4.3 GuessBirthday.java

```
 1  import java.util.Scanner;
 2
 3  public class GuessBirthday {
 4    public static void main(String[] args) {
 5      String set1 =
 6        " 1  3  5  7\n" +
 7        " 9 11 13 15\n" +
 8        "17 19 21 23\n" +
 9        "25 27 29 31";
10
11      String set2 =
12        " 2  3  6  7\n" +
      "10 11 14 15\n" +
      "18 19 22 23\n" +
      "26 27 30 31";

    String set3 =
      " 4  5  6  7\n" +
      "12 13 14 15\n" +
      "20 21 22 23\n" +
      "28 29 30 31";

    String set4 =
      " 8  9 10 11\n" +
      "12 13 14 15\n" +
      "24 25 26 27\n" +
      "28 29 30 31";

    String set5 =
      "16 17 18 19\n" +
      "20 21 22 23\n" +
      "24 25 26 27\n" +
      "28 29 30 31";

```

day to be determined

```
    int day = 0;

    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Prompt the user to answer questions
    System.out.print("Is your birthday in Set1?\n");
    System.out.print(set1);
    System.out.print("\nEnter 0 for No and 1 for Yes: ");
    int answer = input.nextInt();

```

in Set1?

```
    if (answer == 1)
      day += 1;

    // Prompt the user to answer questions
    System.out.print("\nIs your birthday in Set2?\n");
    System.out.print(set2);
    System.out.print("\nEnter 0 for No and 1 for Yes: ");
    answer = input.nextInt();

```

in Set2?

```
    if (answer == 1)
      day += 2;

    // Prompt the user to answer questions
    System.out.print("Is your birthday in Set3?\n");
    System.out.print(set3);
    System.out.print("\nEnter 0 for No and 1 for Yes: ");
    answer = input.nextInt();

```

in Set3?

```
    if (answer == 1)
      day += 4;

    // Prompt the user to answer questions
    System.out.print("\nIs your birthday in Set4?\n");
    System.out.print(set4);
    System.out.print("\nEnter 0 for No and 1 for Yes: ");
    answer = input.nextInt();

    if (answer == 1)
      day += 8;

    // Prompt the user to answer questions
    System.out.print("\nIs your birthday in Set5?\n");
    System.out.print(set5);
    System.out.print("\nEnter 0 for No and 1 for Yes: ");
    answer = input.nextInt();

    if (answer == 1)
      day += 16;

    System.out.println("\nYour birthday is " + day + "!");
  }
}
```

in Set4?

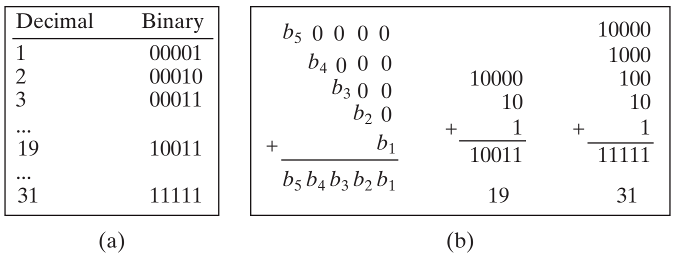
in Set5?

```
Is your birthday in Set1?
 1  3  5  7
 9 11 13 15
17 19 21 23
25 27 29 31
Enter 0 for No and 1 for Yes: 1 ↵Enter

Is your birthday in Set2?
 2  3  6  7
10 11 14 15
18 19 22 23
26 27 30 31
Enter 0 for No and 1 for Yes: 1 ↵Enter

Is your birthday in Set3?
 4  5  6  7
12 13 14 15
20 21 22 23
28 29 30 31
Enter 0 for No and 1 for Yes: 0 ↵Enter

Is your birthday in Set4?
 8  9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 0 ↵Enter

Is your birthday in Set5?
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 1 ↵Enter
Your birthday is 19!
```

line#	day	answer	output
35	0		
44		1	
47	1		
53		1	
56	3		
62		0	
71		0	
80		1	
83	19		
85			Your birthday is 19!

mathematics behind the game

This game is easy to program. You may wonder how the game was created. The mathematics behind the game is actually quite simple. The numbers are not grouped together by accident—the way they are placed in the five sets is deliberate. The starting numbers in the five sets are **1**, **2**, **4**, **8**, and **16**, which correspond to **1**, **10**, **100**, **1000**, and **10000** in binary (binary numbers are introduced in Appendix F, Number Systems). A binary number for decimal integers between **1** and **31** has at most five digits, as shown in Figure 4.3a. Let it be $b_5b_4b_3b_2b_1$. Thus, $b_5b_4b_3b_2b_1 = b_50000 + b_4000 + b_300 + b_20 + b_1$,as shown in Figure 4.3b. If a day's binary number has a digit **1** in b_k, the number should appear in Set*k*. For example, number **19** is binary **10011**, so it appears in Set1, Set2, and Set5. It is binary **1 + 10 + 10000 = 10011** or decimal **1 + 2 + 16 = 19**. Number **31** is binary **11111**, so it appears in Set1, Set2, Set3, Set4, and Set5. It is binary **1 + 10 + 100 + 1000 + 10000 = 11111** or decimal **1 + 2 + 4 + 8 + 16 = 31**.



Decimal	Binary
1	00001
2	00010
3	00011
...	
19	10011
...	
31	11111

(a)

$$\begin{array}{r} b_5\;0\;0\;0\;0 \\ b_4\;0\;0\;0 \\ b_3\;0\;0 \\ b_2\;0 \\ +\qquad b_1 \\ \hline b_5\,b_4\,b_3\,b_2\,b_1 \end{array} \qquad \begin{array}{r} 10000 \\ 10 \\ +\quad 1 \\ \hline 10011 \\ 19 \end{array} \qquad \begin{array}{r} 10000 \\ 1000 \\ 100 \\ 10 \\ +\quad 1 \\ \hline 11111 \\ 31 \end{array}$$

(b)

FIGURE 4.3 (a) A number between **1** and **31** can be represented using a five-digit binary number. (b) A five-digit binary number can be obtained by adding binary numbers **1**, **10**, **100**, **1000**, or **10000**.

Check Point

4.21 If you run Listing 4.3 GuessBirthday.java with input **1** for Set1, Set3, and Set4 and **0** for Set2 and Set5, what will be the birthday?

4.5.2 Case Study: Converting a Hexadecimal Digit to a Decimal Value

The hexadecimal number system has 16 digits: 0–9, A–F. The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15. We now write a program that prompts the user to enter a hex digit and display its corresponding decimal value, as shown in Listing 4.4.

LISTING 4.4 HexDigit2Dec.java

VideoNote
Convert hex to decimal

```
 1 import java.util.Scanner;
 2
 3 public class HexDigit2Dec {
 4   public static void main(String[] args) {
 5     Scanner input = new Scanner(System.in);
 6     System.out.print("Enter a hex digit: ");
 7     String hexString = input.nextLine();
 8
 9     // Check if the hex string has exactly one character
10     if (hexString.length() != 1) {
11       System.out.println("You must enter exactly one character");
12       System.exit(1);
13     }
14
15     // Display decimal value for the hex digit
16     char ch = hexString.charAt(0);
17     if (ch <= 'F' && ch >= 'A') {
18       int value = ch - 'A' + 10;
19       System.out.println("The decimal value for hex digit "
20         + ch + " is " + value);
21     }
22     else if (Character.isDigit(ch)) {
23       System.out.println("The decimal value for hex digit "
24         + ch + " is " + ch);
25     }
26     else {
27       System.out.println(ch + " is an invalid input");
28     }
29   }
30 }
```

input string

check length

is A-F?

is 0-9?

```
Enter a hex digit: AB7C ↵Enter
You must enter exactly one character
```

```
Enter a hex digit: B ↵Enter
The decimal value for hex digit B is 11
```

```
Enter a hex digit: 8 ↵Enter
The decimal value for hex digit 8 is 8
```

```
Enter a hex digit: T ↵Enter
T is an invalid input
```

The program reads a string from the console (line 7) and checks if the string contains a single character (line 10). If not, report an error and exit the program (line 12).

The program invokes the `Character.toUpperCase` method to obtain the character `ch` as an uppercase letter (line 16). If `ch` is between `'A'` and `'F'` (line 17), the corresponding decimal value is `ch – 'A' + 10` (line 18). Note that `ch – 'A'` is `0` if `ch` is `'A'`, `ch – 'A'` is `1` if **ch** is **'B'**, and so on. When two characters perform a numerical operation, the characters' Unicodes are used in the computation.

The program invokes the **Character.isDigit(ch)** method to check if **ch** is between **'0'** and **'9'** (line 22). If so, the corresponding decimal digit is the same as **ch** (lines 23–24).

If **ch** is not between **'A'** and **'F'** nor a digit character, the program displays an error message (line 27).

4.5.3 Case Study: Revising the Lottery Program Using Strings

The lottery program in Listing 3.8, Lottery.java, generates a random two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

1. If the user input matches the lottery number in the exact order, the award is $10,000.
2. If all the digits in the user input match all the digits in the lottery number, the award is $3,000.
3. If one digit in the user input matches a digit in the lottery number, the award is $1,000.

The program in Listing 3.8 uses an integer to store the number. Listing 4.5 gives a new program that generates a random two-digit string instead of a number and receives the user input as a string instead of a number.

LISTING 4.5 LotteryUsingStrings.java

generate a lottery

enter a guess

exact match?

match all digits?

match one digit?

```java
import java.util.Scanner;

public class LotteryUsingStrings {
  public static void main(String[] args) {
    // Generate a lottery as a two-digit string
    String lottery = "" + (int)(Math.random() * 10)
      + (int)(Math.random() * 10);

    // Prompt the user to enter a guess
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your lottery pick (two digits): ");
    String guess = input.nextLine();

    // Get digits from lottery
    char lotteryDigit1 = lottery.charAt(0);
    char lotteryDigit2 = lottery.charAt(1);

    // Get digits from guess
    char guessDigit1 = guess.charAt(0);
    char guessDigit2 = guess.charAt(1);

    System.out.println("The lottery number is " + lottery);

    // Check the guess
    if (guess.equals(lottery))
      System.out.println("Exact match: you win $10,000");
    else if (guessDigit2 == lotteryDigit1
          && guessDigit1 == lotteryDigit2)
      System.out.println("Match all digits: you win $3,000");
    else if (guessDigit1 == lotteryDigit1
          || guessDigit1 == lotteryDigit2
          || guessDigit2 == lotteryDigit1
          || guessDigit2 == lotteryDigit2)
      System.out.println("Match one digit: you win $1,000");
35       else
36         System.out.println("Sorry, no match");
37     }
38   }
```

```
Enter your lottery pick (two digits): 00 ↵Enter
The lottery number is 00
Exact match: you win $10,000
```

```
Enter your lottery pick (two digits): 45 ↵Enter
The lottery number is 54
Match all digits: you win $3,000
```

```
Enter your lottery pick: 23 ↵Enter
The lottery number is 34
Match one digit: you win $1,000
```

```
Enter your lottery pick: 23 ↵Enter
The lottery number is 14
Sorry: no match
```

The program generates two random digits and concatenates them into the string `lottery` (lines 6–7). After this, `lottery` contains two random digits.

The program prompts the user to enter a guess as a two-digit string (line 12) and checks the guess against the lottery number in this order:

- First check whether the guess matches the lottery exactly (line 25).
- If not, check whether the reversal of the guess matches the lottery (line 27).
- If not, check whether one digit is in the lottery (lines 30–33).
- If not, nothing matches and display “Sorry, no match” (line 36).

4.6 Formatting Console Output

Key Point

You can use the `System.out.printf` *method to display formatted output on the console.*

Often, it is desirable to display numbers in a certain format. For example, the following code computes interest, given the amount and the annual interest rate.

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $" + interest);
```

```
Interest is $16.404674
```

Because the interest amount is currency, it is desirable to display only two digits after the decimal point. To do this, you can write the code as follows:

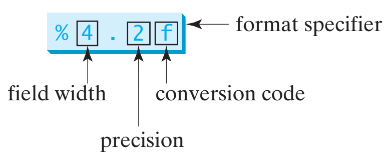
```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $"
  + (int)(interest * 100) / 100.0);
```

```
Interest is $16.4
```

printf

However, the format is still not correct. There should be two digits after the decimal point: `16.40` rather than `16.4`. You can fix it by using the `printf` method, like this:

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.printf("Interest is $%4.2f",
  interest);
```



```
Interest is $16.40
```

The syntax to invoke this method is

```
System.out.printf(format, item1, item2, ..., itemk)
```

where `format` is a string that may consist of substrings and format specifiers.

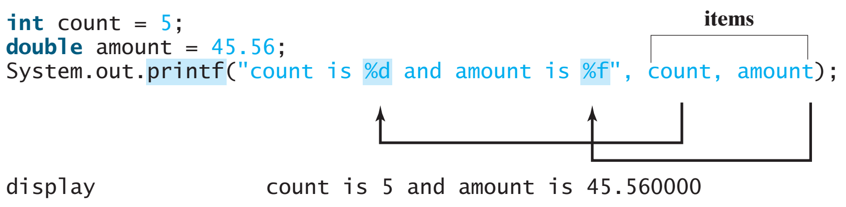
format specifier

A *format specifier* specifies how an item should be displayed. An item may be a numeric value, a character, a Boolean value, or a string. A simple format specifier consists of a percent sign (`%`) followed by a conversion code. Table 4.11 lists some frequently used simple format specifiers.

TABLE 4.11 Frequently Used Format Specifiers

Format Specifier	*Output*	*Example*
`%b`	a Boolean value	true or false
`%c`	a character	'a'
`%d`	a decimal integer	200
`%f`	a floating-point number	45.460000
`%e`	a number in standard scientific notation	4.556000e+01
`%s`	a string	"Java is cool"

Here is an example:



```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

```
display           count is 5 and amount is 45.560000
```

Items must match the format specifiers in order, in number, and in exact type. For example, the format specifier for `count` is `%d` and for `amount` is `%f`. By default, a floating-point value is displayed with six digits after the decimal point. You can specify the width and precision in a format specifier, as shown in the examples in Table 4.12.

TABLE 4.12 Examples of Specifying Width and Precision

Example	*Output*
`%5c`	Output the character and add four spaces before the character item, because the width is 5.
`%6b`	Output the Boolean value and add one space before the false value and two spaces before the true value.
`%5d`	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
`%10.2f`	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased.
`%10.2e`	Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
`%12s`	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

If an item requires more spaces than the specified width, the width is automatically increased. For example, the following code

```
System.out.printf("%3d#%2s#%4.2f\n", 1234, "Java", 51.6653);
```

displays

```
1234#Java#51.67
```

The specified width for `int` item `1234` is `3`, which is smaller than its actual size `4`. The width is automatically increased to `4`. The specified width for string item `Java` is `2`, which is smaller than its actual size `4`. The width is automatically increased to `4`. The specified width for `double` item `51.6653` is `4`, but it needs width 5 to display 51.67, so the width is automatically increased to `5`.

By default, the output is right justified. You can put the minus sign (`-`) in the format specifier to specify that the item is left justified in the output within the specified field. For example, the following statements

right justify
left justify

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.63);
System.out.printf("%-8d%-8s%-8.1f \n", 1234, "Java", 5.63);
```

display

```
|←  8  →|←  8  →|←  8  →|
□□□□1234□□□□Java□□□□□5.6
1234□□□□Java□□□□5.6□□□□□
```

where the square box (□) denotes a blank space.

Caution

The items must match the format specifiers in exact type. The item for the format specifier `%f` or `%e` must be a floating-point type value such as `40.0`, not `40`. Thus, an `int` variable cannot match `%f` or `%e`.

Tip

The `%` sign denotes a format specifier. To output a literal `%` in the format string, use `%%`.

Listing 4.6 gives a program that uses `printf` to display a table.

LISTING 4.6 FormatDemo.java

display table header

values for 30 degrees

values for 60 degrees

```java
 1  public class FormatDemo {
 2    public static void main(String[] args) {
 3      // Display the header of the table
 4      System.out.printf("%-10s%-10s%-10s%-10s%-10s\n", "Degrees",
 5        "Radians", "Sine", "Cosine", "Tangent");
 6
 7      // Display values for 30 degrees
 8      int degrees = 30;
 9      double radians = Math.toRadians(degrees);
10      System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
11        radians, Math.sin(radians), Math.cos(radians),
12        Math.tan(radians));
13
14      // Display values for 60 degrees
15      degrees = 60;
16      radians = Math.toRadians(degrees);
17      System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,
18        radians, Math.sin(radians), Math.cos(radians),
19        Math.tan(radians));
20    }
21  }
```

```
Degrees   Radians   Sine      Cosine    Tangent
30        0.5236    0.5000    0.8660    0.5773
60        1.0472    0.8660    0.5000    1.7320
```

The statement in lines 4–5 displays the column names of the table. The column names are strings. Each string is displayed using the specifier `%-10s`, which left-justifies the string. The statement in lines 10–12 displays the degrees as an integer and four float values. The integer is displayed using the specifier `%-10d` and each float is displayed using the specifier `%-10.4f`, which specifies four digits after the decimal point.

Check Point

4.22 What are the format specifiers for outputting a Boolean value, a character, a decimal integer, a floating-point number, and a string?

4.23 What is wrong in the following statements?

(a) `System.out.printf("%5d %d", 1, 2, 3);`

(b) `System.out.printf("%5d %f", 1);`

(c) `System.out.printf("%5d %f", 1, 2);`