CHAPTER 3

SELECTIONS

Objectives

- To declare `boolean` type and write Boolean expressions using comparison operators (§3.2).
- To program `AdditionQuiz` using Boolean expressions (§3.3).
- To implement selection control using one-way `if` statements (§3.4)
- To program the `GuessBirthday` game using one-way `if` statements (§3.5).
- To implement selection control using two-way `if` statements (§3.6).
- To implement selection control using nested `if` statements (§3.7).
- To avoid common errors in `if` statements (§3.8).
- To program using selection statements for a variety of examples(`SubtractionQuiz`, `BMI`, `ComputeTax`) (§3.9–3.11).
- To generate random numbers using the `Math.random()` method (§3.9).
- To combine conditions using logical operators (`&&`, `||`, and `!`) (§3.12).
- To program using selection statements with combined conditions (`LeapYear`, `Lottery`) (§§3.13–3.14).
- To implement selection control using `switch` statements (§3.15).
- To write expressions using the conditional operator (§3.16).
- To format output using the `System.out.printf` method and to format strings using the `String.format` method (§3.17).
- To examine the rules governing operator precedence and associativity (§3.18).
- (GUI) To get user confirmation using confirmation dialogs (§3.19).

3.1 Introduction

problem

If you enter a negative value for `radius` in Listing 2.2, ComputeAreaWithConsoleInput.java, the program prints an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?

Like all high-level programming languages, Java provides selection statements that let you choose actions with two or more alternative courses. You can use the following selection statement to replace lines 12–17 in Listing 2.2:

```
if (radius < 0)
  System.out.println("Incorrect input");
else {
  area = radius * radius * 3.14159;
  System.out.println("Area is " + area);
}
```

Selection statements use conditions. Conditions are Boolean expressions. This chapter first introduces Boolean types, values, comparison operators, and expressions.

3.2 `boolean` Data Type

comparison operators

How do you compare two values, such as whether a radius is greater than `0`, equal to `0`, or less than `0`? Java provides six *comparison operators* (also known as *relational operators*), shown in Table 3.1, which can be used to compare two values (assume radius is `5` in the table).

TABLE 3.1 Comparison Operators

Operator	*Name*	*Example*	*Result*
`<`	less than	`radius < 0`	`false`
`<=`	less than or equal to	`radius <= 0`	`false`
`>`	greater than	`radius > 0`	`true`
`>=`	greater than or equal to	`radius >= 0`	`true`
`==`	equal to	`radius == 0`	`false`
`!=`	not equal to	`radius != 0`	`true`

compare characters

Note

You can also compare characters. Comparing characters is the same as comparing their Unicodes. For example, `'a'` is larger than `'A'` because the Unicode of `'a'` is larger than the Unicode of `'A'`. See Appendix B, "The ASCII Character Sets," to find the order of characters.

== vs. =

Caution

The equality comparison operator is two equal signs (`==`), not a single equal sign (`=`). The latter symbol is for assignment.

The result of the comparison is a Boolean value: `true` or `false`. For example, the following statement displays `true`:

```
double radius = 1;
System.out.println(radius > 0);
```

Boolean variable

A variable that holds a Boolean value is known as a *Boolean variable*. The `boolean` data type is used to declare Boolean variables. A `boolean` variable can hold one of the two values:

`true` and `false`. For example, the following statement assigns `true` to the variable `lightsOn`:

```
boolean lightsOn = true;
```

`true` and `false` are literals, just like a number such as `10`. They are reserved words and cannot be used as identifiers in your program.

Boolean literals

3.3 Problem: A Simple Math Learning Tool

Video Note
Program addition quiz

Suppose you want to develop a program to let a first-grader practice addition. The program randomly generates two single-digit integers, `number1` and `number2`, and displays to the student a question such as "What is 7 + 9?", as shown in the sample run. After the student types the answer, the program displays a message to indicate whether it is true or false.

There are several ways to generate random numbers. For now, generate the first integer using `System.currentTimeMillis() % 10` and the second using `System.currentTimeMillis() * 7 % 10`. Listing 3.1 gives the program. Lines 5–6 generate two numbers, `number1` and `number2`. Line 14 obtains an answer from the user. The answer is graded in line 18 using a Boolean expression `number1 + number2 == answer`.

LISTING 3.1 AdditionQuiz.java

```
1 import java.util.Scanner;
2
3 public class AdditionQuiz {
4   public static void main(String[] args) {
5     int number1 = (int)(System.currentTimeMillis() % 10);                    generate number1
6     int number2 = (int)(System.currentTimeMillis() * 7 % 10);                generate number2
7
8     // Create a Scanner
9     Scanner input = new Scanner(System.in);
10
11    System.out.print(                                                        show question
12      "What is " + number1 + " + " + number2 + "? ");
13
14    int answer = input.nextInt();
15
16    System.out.println(                                                      display result
17      number1 + " + " + number2 + " = " + answer + " is " +
18      (number1 + number2 == answer));
19  }
20 }
```

```
What is 1 + 7? 8 Enter
1 + 7 = 8 is true
```

```
What is 4 + 8? 9 Enter
4 + 8 = 9 is false
```

line#	number1	number2	answer	output
5	4			
6		8		
14			9	
16				4 + 8 = 9 is false

3.4 `if` Statements

why `if` statement?

The preceding program displays a message such as "6 + 2 = 7 is false." If you wish the message to be "6 + 2 = 7 is incorrect," you have to use a selection statement to carry out this minor change.

This section introduces selection statements. Java has several types of selection statements: one-way `if` statements, two-way `if` statements, nested `if` statements, `switch` statements, and conditional expressions.

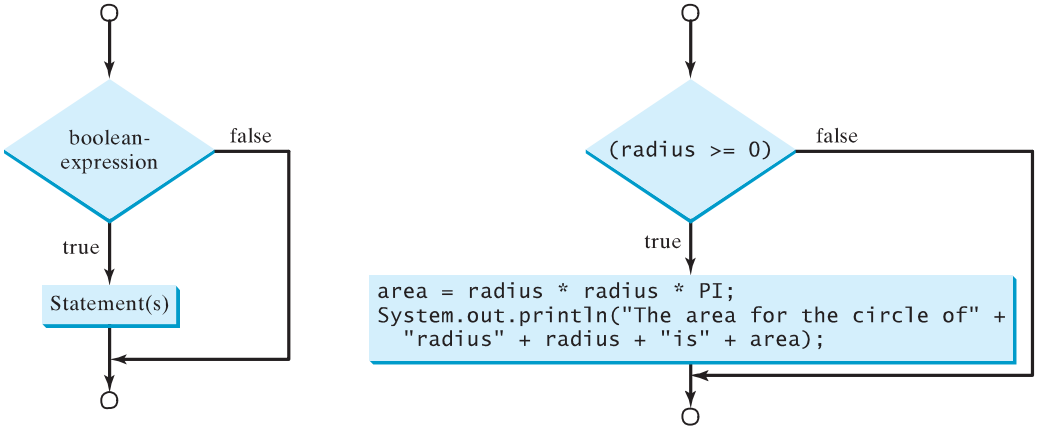
3.4.1 One-Way `if` Statements

A one-way `if` statement executes an action if and only if the condition is `true`. The syntax for a one-way `if` statement is shown below:

`if` statement

```
if (boolean-expression) {
  statement(s);
}
```

The execution flow chart is shown in Figure 3.1(a).



(a) (b)

FIGURE 3.1 An `if` statement executes statements if the `boolean-expression` evaluates to `true`.

If the `boolean-expression` evaluates to `true`, the statements in the block are executed. As an example, see the following code:

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
    radius + " is " + area);
}
```

The flow chart of the preceding statement is shown in Figure 3.1(b). If the value of `radius` is greater than or equal to `0`, then the `area` is computed and the result is displayed; otherwise, the two statements in the block will not be executed.

The `boolean-expression` is enclosed in parentheses. For example, the code in (a) below is wrong. It should be corrected, as shown in (b).

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

The block braces can be omitted if they enclose a single statement. For example, the following statements are equivalent.

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

Listing 3.2 gives a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

LISTING 3.2 SimpleIfDemo.java

```
1 import java.util.Scanner;
2
3 public class SimpleIfDemo {
4   public static void main(String[] args) {
5     Scanner input = new Scanner(System.in);
6     System.out.println("Enter an integer: ");
7     int number = input.nextInt();                    enter input
8
9     if (number % 5 == 0)                             check 5
10      System.out.println("HiFive");
11
12    if (number % 2 == 0)                             check even
13      System.out.println("HiEven");
14  }
15 }
```

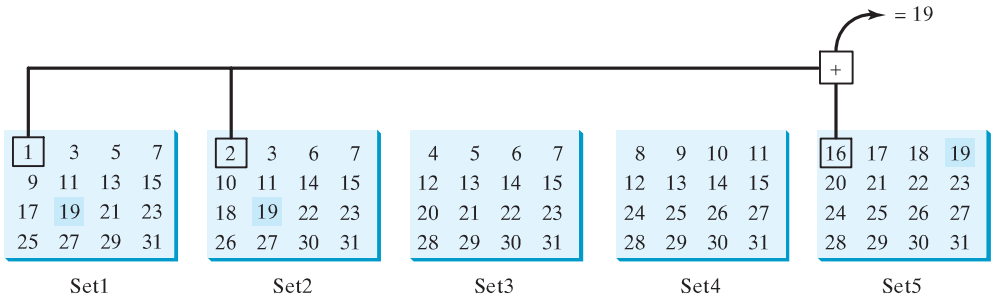
```
Enter an integer: 4 Enter
HiEven
```

```
Enter an integer: 30 Enter
HiFive
HiEven
```

The program prompts the user to enter an integer (line 7) and displays HiFive if it is divisible by 5 (lines 9–10) and HiEven if it is divisible by 2 (lines 12–13).

3.5 Problem: 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 3.3 gives a program that prompts the user to answer whether the day is in Set1 (lines 41–47), in Set2 (lines 50–56), in Set3 (lines 59–65), in Set4 (lines 68–74), and in Set5 (lines 77–83). 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 3.3 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" +
13      "10 11 14 15\n" +
14      "18 19 22 23\n" +
15      "26 27 30 31";
16
17    String set3 =
18      " 4  5  6  7\n" +
19      "12 13 14 15\n" +
20      "20 21 22 23\n" +
21      "28 29 30 31";
22
23    String set4 =
24      " 8  9 10 11\n" +
25      "12 13 14 15\n" +
26      "24 25 26 27\n" +
27      "28 29 30 31";
28
29    String set5 =
30      "16 17 18 19\n" +
31      "20 21 22 23\n" +
32      "24 25 26 27\n" +
33      "28 29 30 31";
34
35    int day = 0;
36
37    // Create a Scanner
38    Scanner input = new Scanner(System.in);
39
40    // Prompt the user to answer questions
41    System.out.print("Is your birthday in Set1?\n");
42    System.out.print(set1);
43    System.out.print("\nEnter 0 for No and 1 for Yes: ");
44    int answer = input.nextInt();
45
46    if (answer == 1)
47      day += 1;
48
```

day to be determined

in Set1?

```
49      // Prompt the user to answer questions
50      System.out.print("\nIs your birthday in Set2?\n" );
51      System.out.print(set2);
52      System.out.print("\nEnter 0 for No and 1 for Yes: ");
53      answer = input.nextInt();
54
55      if (answer == 1)                                          in Set2?
56        day += 2;
57
58      // Prompt the user to answer questions
59      System.out.print("Is your birthday in Set3?\n");
60      System.out.print(set3);
61      System.out.print("\nEnter 0 for No and 1 for Yes: ");
62      answer = input.nextInt();
63
64      if (answer == 1)                                          in Set3?
65        day += 4;
66
67      // Prompt the user to answer questions
68      System.out.print("\nIs your birthday in Set4?\n");
69      System.out.print(set4);
70      System.out.print("\nEnter 0 for No and 1 for Yes: ");
71      answer = input.nextInt();
72
73      if (answer == 1)                                          in Set4?
74        day += 8;
75
76      // Prompt the user to answer questions
77      System.out.print("\nIs your birthday in Set5?\n");
78      System.out.print(set5);
79      System.out.print("\nEnter 0 for No and 1 for Yes: ");
80      answer = input.nextInt();
81
82      if (answer == 1)                                          in Set5?
83        day += 16;
84
85      System.out.println("\nYour birthday is " + day + "!");
86    }
87 }
```

```
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		
			Your birthday is 19

mathematics behind the game

The 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. A binary number for decimal integers between 1 and 31 has at most five digits, as shown in Figure 3.2(a). Let it be $b_5b_4b_3b_2b_1$. So, $b_5b_4b_3b_2b_1 = b_5\,0000 + b_4\,000 + b_3\,00 + b_2\,0 + b_1$, as shown in Figure 3.2(b). 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)

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

(b)

FIGURE 3.2 (a) A number between 1 and 31 can be represented using a 5-digit binary number. (b) A 5-digit binary number can be obtained by adding binary numbers 1, 10, 100, 1000, or 10000.

3.6 Two-Way if Statements

A one-way `if` statement takes an action if the specified condition is `true`. If the condition is `false`, nothing is done. But what if you want to take alternative actions when the condition is `false`? You can use a two-way `if` statement. The actions that a two-way `if` statement specifies differ based on whether the condition is `true` or `false`.

Here is the syntax for a two-way `if` statement:

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

The flow chart of the statement is shown in Figure 3.3.

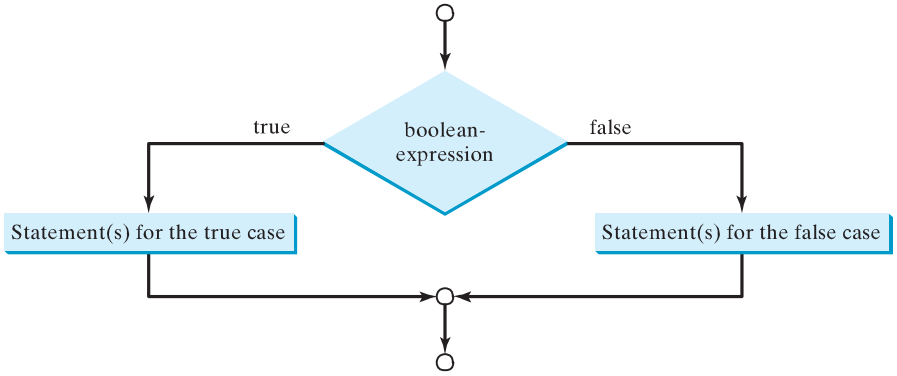


FIGURE 3.3 An `if . . . else` statement executes statements for the true case if the `boolean-expression` evaluates to `true`; otherwise, statements for the false case are executed.

If the `boolean-expression` evaluates to `true`, the statement(s) for the true case are executed; otherwise, the statement(s) for the false case are executed. For example, consider the following code:

two-way `if` statement

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
    radius + " is " + area);
}
else {
  System.out.println("Negative input");
}
```

If `radius >= 0` is `true`, `area` is computed and displayed; if it is `false`, the message `"Negative input"` is printed.

As usual, the braces can be omitted if there is only one statement within them. The braces enclosing the `System.out.println("Negative input")` statement can therefore be omitted in the preceding example.

Here is another example of using the `if ... else` statement. The example checks whether a number is even or odd, as follows:

```
if (number % 2 == 0)
  System.out.println(number + " is even.");
else
  System.out.println(number + " is odd.");
```

3.7 Nested `if` Statements

The statement in an `if` or `if ... else` statement can be any legal Java statement, including another `if` or `if ... else` statement. The inner `if` statement is said to be *nested* inside the outer `if` statement. The inner `if` statement can contain another `if` statement; in fact, there is no limit to the depth of the nesting. For example, the following is a nested `if` statement:

nested `if` statement

```
if (i > k) {
  if (j > k)
    System.out.println("i and j are greater than k");
}
else
  System.out.println("i is less than or equal to k");
```

The `if (j > k)` statement is nested inside the `if (i > k)` statement.

The nested `if` statement can be used to implement multiple alternatives. The statement given in Figure 3.4(a), for instance, assigns a letter grade to the variable `grade` according to the score, with multiple alternatives.

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

(a)

Equivalent

This is better

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

(b)

FIGURE 3.4 A preferred format for multiple alternative `if` statements is shown in (b).

The execution of this `if` statement proceeds as follows. The first condition `(score >= 90.0)` is tested. If it is `true`, the grade becomes `'A'`. If it is `false`, the second condition `(score >= 80.0)` is tested. If the second condition is `true`, the grade becomes `'B'`. If that condition is `false`, the third condition and the rest of the conditions (if necessary) continue to be tested until a condition is met or all of the conditions prove to be `false`. If all of the conditions are `false`, the grade becomes `'F'`. Note that a condition is tested only when all of the conditions that come before it are `false`.

The `if` statement in Figure 3.4(a) is equivalent to the `if` statement in Figure 3.4(b). In fact, Figure 3.4(b) is the preferred writing style for multiple alternative `if` statements. This style avoids deep indentation and makes the program easy to read.

Tip

Often, to assign a test condition to a **boolean** variable, new programmers write code as in (a) below:

assign **boolean** variable

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

This is shorter

```
boolean even
  = number % 2 == 0;
```

(b)

The code can be simplified by assigning the test value directly to the variable, as shown in (b).

3.8 Common Errors in Selection Statements

The following errors are common among new programmers.

Common Error 1: Forgetting Necessary Braces

The braces can be omitted if the block contains a single statement. However, forgetting the braces when they are needed for grouping multiple statements is a common programming error. If you modify the code by adding new statements in an **if** statement without braces, you will have to insert the braces. For example, the code in (a) below is wrong. It should be written with braces to group multiple statements, as shown in (b).

```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

(a) Wrong

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b) Correct

Common Error 2: Wrong Semicolon at the if Line

Adding a semicolon at the **if** line, as shown in (a) below, is a common mistake.

Logic Error

```
if (radius >= 0);
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(a)

Equivalent

Empty Block

```
if (radius >= 0) { };
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b)

This mistake is hard to find, because it is neither a compilation error nor a runtime error; it is a logic error. The code in (a) is equivalent to that in (b) with an empty block.

This error often occurs when you use the next-line block style. Using the end-of-line block style can help prevent the error.

Common Error 3: Redundant Testing of Boolean Values

To test whether a **boolean** variable is **true** or **false** in a test condition, it is redundant to use the equality comparison operator like the code in (a):

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

This is better

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

Instead, it is better to test the `boolean` variable directly, as shown in (b). Another good reason for doing this is to avoid errors that are difficult to detect. Using the `=` operator instead of the `==` operator to compare equality of two items in a test condition is a common error. It could lead to the following erroneous statement:

```
if (even = true)
  System.out.println("It is even.");
```

This statement does not have syntax errors. It assigns `true` to `even`, so that `even` is always `true`.

Common Error 4: Dangling `else` Ambiguity

The code in (a) below has two `if` clauses and one `else` clause. Which `if` clause is matched by the `else` clause? The indentation indicates that the `else` clause matches the first `if` clause. However, the `else` clause actually matches the second `if` clause. This situation is known as the *dangling-else ambiguity*. The `else` clause always matches the most recent unmatched `if` clause in the same block. So, the statement in (a) is equivalent to the code in (b).

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

This is better with correct indentation

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Since `(i > j)` is false, nothing is printed from the statement in (a) and (b). To force the `else` clause to match the first `if` clause, you must add a pair of braces:

```
int i = 1, j = 2, k = 3;

if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

This statement prints `B`.

3.9 Problem: An Improved Math Learning Tool

Video Note
Program subtraction quiz

Suppose you want to develop a program for a first-grader to practice subtraction. The program randomly generates two single-digit integers, `number1` and `number2`, with `number1 >= number2` and displays to the student a question such as "What is 9 − 2?" After the student enters the answer, the program displays a message indicating whether it is correct.

`random()` method

The previous programs generate random numbers using `System.currentTimeMillis()`. A better approach is to use the `random()` method in the `Math` class. Invoking this method returns a random double value `d` such that $0.0 \le d < 1.0$. So, `(int)(Math.random() * 10)` returns a random single-digit integer (i.e., a number between 0 and 9).

The program may work as follows:

- Generate two single-digit integers into `number1` and `number2`.
- If `number1 < number2`, swap `number1` with `number2`.

- Prompt the student to answer "What is number1 – number2?"
- Check the student's answer and display whether the answer is correct.

The complete program is shown in Listing 3.4.

LISTING 3.4 SubtractionQuiz.java

```
1 import java.util.Scanner;
2
3 public class SubtractionQuiz {
4   public static void main(String[] args) {
5     // 1. Generate two random single-digit integers
6     int number1 = (int)(Math.random() * 10);                    random numbers
7     int number2 = (int)(Math.random() * 10);
8
9     // 2. If number1 < number2, swap number1 with number2
10    if (number1 < number2) {
11      int temp = number1;
12      number1 = number2;
13      number2 = temp;
14    }
15
16    // 3. Prompt the student to answer "What is number1 - number2?"
17    System.out.print
18      ("What is " + number1 + " - " + number2 + "? ");
19    Scanner input = new Scanner(System.in);
20    int answer = input.nextInt();                               get answer
21
22    // 4. Grade the answer and display the result
23    if (number1 - number2 == answer)                            check the answer
24      System.out.println("You are correct!");
25    else
26      System.out.println("Your answer is wrong\n" + number1 + " - "
27        + number2 + " should be " + (number1 - number2));
28  }
29 }
```

```
What is 6 - 6? 0 Enter
You are correct!
```

```
What is 9 - 2? 5 Enter
Your answer is wrong
9 - 2 should be 7
```

line#	number1	number2	temp	answer	output
6	2				
7		9			
11			2		
12	9				
13		2			
20				5	
26					Your answer is wrong 9 – 2 should be 7

To swap two variables `number1` and `number2`, a temporary variable `temp` (line 11) is used to first hold the value in `number1`. The value in `number2` is assigned to `number1` (line 12), and the value in `temp` is assigned to `number2` (line 13).

3.10 Problem: Computing Body Mass Index

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
below 16	seriously underweight
16–18	underweight
18–24	normal weight
24–29	overweight
29–35	seriously overweight
above 35	gravely overweight

Write a program that prompts the user to enter a weight in pounds and height in inches and display the BMI. Note that one pound is `0.45359237` kilograms and one inch is `0.0254` meters. Listing 3.5 gives the program.

LISTING 3.5 ComputeBMI.java

```
1  import java.util.Scanner;
2
3  public class ComputeAndInterpretBMI {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6
7      // Prompt the user to enter weight in pounds
8      System.out.print("Enter weight in pounds: ");
9      double weight = input.nextDouble();                    // input weight
10
11     // Prompt the user to enter height in inches
12     System.out.print("Enter height in inches: ");
13     double height = input.nextDouble();                    // input height
14
15     final double KILOGRAMS_PER_POUND = 0.45359237; // Constant
16     final double METERS_PER_INCH = 0.0254; // Constant
17
18     // Compute BMI
19     double weightInKilograms = weight * KILOGRAMS_PER_POUND;
20     double heightInMeters = height * METERS_PER_INCH;
21     double bmi = weightInKilograms /                       // compute bmi
22       (heightInMeters * heightInMeters);
23
24     // Display result
25     System.out.printf("Your BMI is %5.2f\n", bmi);         // display output
26     if (bmi < 16)
27       System.out.println("You are seriously underweight");
28     else if (bmi < 18)
29       System.out.println("You are underweight");
30     else if (bmi < 24)
31       System.out.println("You are normal weight");
```

```
32      else if (bmi < 29)
33        System.out.println("You are overweight");
34      else if (bmi < 35)
35        System.out.println("You are seriously overweight");
36      else
37        System.out.println("You are gravely overweight");
38    }
39 }
```

```
Enter weight in pounds: 146 Enter
Enter height in inches: 70 Enter
Your BMI is 20.948603801493316
You are normal weight
```

line#	weight	height	WeightInKilograms	heightInMeters	bmi	output
9	146					
13		70				
19			66.22448602			
20				1.778		
21					20.9486	
25						Your BMI is 20.95
31						You are normal weight

Two constants `KILOGRAMS_PER_POUND` and `METERS_PER_INCH` are defined in lines 15–16. Using constants here makes programs easy to read.

3.11 Problem: Computing Taxes

Video Note
Use multiple alternative if statements

The United States federal personal income tax is calculated based on filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates vary every year. Table 3.2 shows the rates for 2009. If you are, say, single with a taxable income of $10,000, the first $8,350 is taxed at 10% and the other $1,650 is taxed at 15%. So, your tax is $1,082.5

TABLE 3.2 2009 U.S. Federal Personal Tax Rates

Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualified Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,525 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

You are to write a program to compute personal income tax. Your program should prompt the user to enter the filing status and taxable income and compute the tax. Enter 0 for single filers, 1 for married filing jointly, 2 for married filing separately, and 3 for head of household.

Your program computes the tax for the taxable income based on the filing status. The filing status can be determined using `if` statements outlined as follows:

```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married filing jointly
}
else if (status == 2) {
  // Compute tax for married filing separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

For each filing status there are six tax rates. Each rate is applied to a certain amount of taxable income. For example, of a taxable income of $400,000 for single filers, $8,350 is taxed at 10%, (33,950 − 8,350) at 15%, (82,250 − 33,950) at 25%, (171,550 − 82,250) at 28%, (372,950 − 171,550) at 33%, and (400,000 − 372,950) at 35%.

Listing 3.6 gives the solution to compute taxes for single filers. The complete solution is left as an exercise.

LISTING 3.6 ComputeTax.java

```
1 import java.util.Scanner;
2
3 public class ComputeTax {
4   public static void main(String[] args) {
5     // Create a Scanner
6     Scanner input = new Scanner(System.in);
7
8     // Prompt the user to enter filing status
9     System.out.print(
10      "(0-single filer, 1-married jointly,\n" +
11      "2-married separately, 3-head of household)\n" +
12      "Enter the filing status: ");
13    int status = input.nextInt();
14
15    // Prompt the user to enter taxable income
16    System.out.print("Enter the taxable income: ");
17    double income = input.nextDouble();
18
19    // Compute tax
20    double tax = 0;
21
22    if (status == 0) { // Compute tax for single filers
23      if (income <= 8350)
24        tax = income * 0.10;
25      else if (income <= 33950)
26        tax = 8350 * 0.10 + (income - 8350) * 0.15;
27      else if (income <= 82250)
28        tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
```

input status

input income

compute tax

```
29            (income - 33950) * 0.25;
30        else if (income <= 171550)
31          tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
32            (82250 - 33950) * 0.25 + (income - 82250) * 0.28;
33        else if (income <= 372950)
34          tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
35            (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
36            (income - 171550) * 0.35;
37        else
38          tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
39            (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
40            (372950 - 171550) * 0.33 + (income - 372950) * 0.35;
41      }
42      else if (status == 1) { // Compute tax for married file jointly
43        // Left as exercise
44      }
45      else if (status == 2) { // Compute tax for married separately
46        // Left as exercise
47      }
48      else if (status == 3) { // Compute tax for head of household
49        // Left as exercise
50      }
51      else {
52        System.out.println("Error: invalid status");
53        System.exit(0);                                          exit program
54      }
55
56      // Display the result
57      System.out.println("Tax is " + (int)(tax * 100) / 100.0);  display output
58    }
59 }
```

```
(0-single filer, 1-married jointly,
2-married separately, 3-head of household)
Enter the filing status: 0 Enter
Enter the taxable income: 400000 Enter
Tax is 117683.5
```

line#	status	income	tax	output
13	0			
17		400000		
20			0	
38			117683.5	
57				Tax is 117683.5

The program receives the filing status and taxable income. The multiple alternative **if** statements (lines 22, 42, 45, 48, 51) check the filing status and compute the tax based on the filing status.

System.exit(0) (line 53) is defined in the **System** class. Invoking this method terminates the program. The argument **0** indicates that the program is terminated normally. **System.exit(0)**

An initial value of **0** is assigned to **tax** (line 20). A syntax error would occur if it had no initial value, because all of the other statements that assign values to **tax** are within the **if**

statement. The compiler thinks that these statements may not be executed and therefore reports a syntax error.

test all cases

To test a program, you should provide the input that covers all cases. For this program, your input should cover all statuses (0, 1, 2, 3). For each status, test the tax for each of the six brackets. So, there are a total of 24 cases.

Tip

incremental development and testing

For all programs, you should write a small amount of code and test it before moving on to add more code. This is called *incremental development and testing*. This approach makes debugging easier, because the errors are likely in the new code you just added.

3.12 Logical Operators

Sometimes, whether a statement is executed is determined by a combination of several conditions. You can use logical operators to combine these conditions. *Logical operators*, also known as *Boolean operators*, operate on Boolean values to create a new Boolean value. Table 3.3 gives a list of Boolean operators. Table 3.4 defines the not (`!`) operator. The not (`!`) operator negates `true` to `false` and `false` to `true`. Table 3.5 defines the and (`&&`) operator. The and (`&&`) of two Boolean operands is `true` if and only if both operands are `true`. Table 3.6 defines the or (`||`) operator. The or (`||`) of two Boolean operands is `true` if at least one of the operands is `true`. Table 3.7 defines the exclusive or (`^`) operator. The exclusive or (`^`) of two Boolean operands is `true` if and only if the two operands have different Boolean values.

TABLE 3.3 Boolean Operators

Operator	*Name*	*Description*
`!`	not	logical negation
`&&`	and	logical conjunction
`||`	or	logical disjunction
`^`	exclusive or	logical exclusion

TABLE 3.4 Truth Table for Operator `!`

p	*!p*	*Example (assume age = 24, gender = 'F')*
`true`	`false`	`!(age > 18)` is `false`, because `(age > 18)` is `true`.
`false`	`true`	`!(gender == 'M')` is `true`, because `(gender == 'M')` is `false`.

TABLE 3.5 Truth Table for Operator `&&`

p1	*p2*	*p1 && p2*	*Example (assume* age = 24, gender = 'F')
`false`	`false`	`false`	`(age > 18) && (gender == 'F')` is `true`, because `(age > 18)` and `(gender == 'F')` are both `true`.
`false`	`true`	`false`	
`true`	`false`	`false`	`(age > 18) && (gender != 'F')` is `false`, because `(gender != 'F')` is `false`.
`true`	`true`	`true`	

TABLE 3.6 Truth Table for Operator ||

p1	*p2*	*p1* || *p2*	*Example (assume* age = 24, gender = 'F'*)*
false	false	false	`(age > 34) || (gender == 'F')` is `true`, because `(gender == 'F')` is `true`.
false	true	true	
true	false	true	`(age > 34) || (gender == 'M')` is `false`, because `(age > 34)` and `(gender == 'M')` are both `false`.
true	true	true	

TABLE 3.7 Truth Table for Operator ^

p1	*p2*	*p1* ^ *p2*	*Example (assume* age = 24, gender = 'F'*)*
false	false	false	`(age > 34) ^ (gender == 'F')` is `true`, because `(age > 34)` is `false` but `(gender == 'F')` is `true`.
false	true	true	
true	false	true	`(age > 34) || (gender == 'M')` is `false`, because `(age > 34)` and `(gender == 'M')` are both `false`.
true	true	false	

Listing 3.7 gives a program that checks whether a number is divisible by 2 and 3, by 2 or 3, and by 2 or 3 but not both:

LISTING 3.7 TestBooleanOperators.java

```
1 import java.util.Scanner;                                          import class
2
3 public class TestBooleanOperators {
4   public static void main(String[] args) {
5     // Create a Scanner
6     Scanner input = new Scanner(System.in);
7
8     // Receive an input
9     System.out.print("Enter an integer: ");
10    int number = input.nextInt();                                  input
11
12    System.out.println("Is " + number +
13      "\n\tdivisible by 2 and 3? " +
14      (number % 2 == 0 && number % 3 == 0 )                        and
15      + "\n\tdivisible by 2 or 3? " +
16      (number % 2 == 0 || number % 3 == 0 ) +                      or
17      "\n\tdivisible by 2 or 3, but not both? "
18      + (number % 2 == 0 ^ number % 3 == 0 ));                     exclusive or
19  }
20 }
```

```
Enter an integer: 18 Enter
Is 18
  divisible by 2 and 3? true
  divisible by 2 or 3? true
  divisible by 2 or 3, but not both? false
```

A long string is formed by concatenating the substrings in lines 12–18. The three `\n` characters display the string in four lines. `(number % 2 == 0 && number % 3 == 0)` (line 14) checks whether the number is divisible by `2` and `3`. `(number % 2 == 0 || number % 3 == 0)` (line 16) checks whether the number is divisible by `2` or `3`. `(number % 2 == 0 ^ number % 3 == 0)` (line 20) checks whether the number is divisible by `2` or `3`, but not both.

Caution

In mathematics, the expression

```
1 <= numberOfDaysInAMonth <= 31
```

is correct. However, it is incorrect in Java, because `1 <= numberOfDaysInAMonth` is evaluated to a `boolean` value, which cannot be compared with `31`. Here, two operands (a `boolean` value and a numeric value) are *incompatible*. The correct expression in Java is

incompatible operands

```
(1 <= numberOfDaysInAMonth) && (numberOfDaysInAMonth <= 31)
```

Note

As shown in the preceding chapter, a `char` value can be cast into an `int` value, and vice versa. A `boolean` value, however, cannot be cast into a value of another type, nor can a value of another type be cast into a `boolean` value.

cannot cast **boolean**

Note

De Morgan's law, named after Indian-born British mathematician and logician Augustus De Morgan (1806–1871), can be used to simplify Boolean expressions. The law states

De Morgan's law

```
!(condition1 && condition2) is same as !condition1 || !condition2
!(condition1 || condition2) is same as !condition1 && !condition2
```

For example,

```
!(n == 2 || n == 3) is same as n != 2 && n != 3
!(n % 2 == 0 && n % 3 == 0) is same as n % 2 != 0 || n % 3 != 0
```

If one of the operands of an `&&` operator is `false`, the expression is `false`; if one of the operands of an `||` operator is `true`, the expression is `true`. Java uses these properties to improve the performance of these operators. When evaluating `p1 && p2`, Java first evaluates `p1` and then, if `p1` is `true`, evaluates `p2`; if `p1` is `false`, it does not evaluate `p2`. When evaluating `p1 || p2`, Java first evaluates `p1` and then, if `p1` is `false`, evaluates `p2`; if `p1` is `true`, it does not evaluate `p2`. Therefore, `&&` is referred to as the *conditional* or *short-circuit AND* operator, and `||` is referred to as the *conditional* or *short-circuit OR* operator.

conditional operator
short-circuit operator

3.13 Problem: Determining Leap Year

A year is a *leap year* if it is divisible by `4` but not by `100` or if it is divisible by `400`. So you can use the following Boolean expressions to check whether a year is a leap year:

leap year

```
// A leap year is divisible by 4
boolean isLeapYear = (year % 4 == 0);

// A leap year is divisible by 4 but not by 100
isLeapYear = isLeapYear && (year % 100 != 0);

// A leap year is divisible by 4 but not by 100 or divisible by 400
isLeapYear = isLeapYear || (year % 400 == 0);
```

or you can combine all these expressions into one like this:

```
isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
```

Listing 3.8 gives the program that lets the user enter a year and checks whether it is a leap year.

LISTING 3.8 LeapYear.java

```
1 import java.util.Scanner;
2
3 public class LeapYear {
4   public static void main(String[] args) {
5     // Create a Scanner
6     Scanner input = new Scanner(System.in);
7     System.out.print("Enter a year: ");
8     int year = input.nextInt();                                    input
9
10    // Check if the year is a leap year
11    boolean isLeapYear =                                           leap year?
12      (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
13
14    // Display the result
15    System.out.println(year + " is a leap year? " + isLeapYear);   display result
16  }
17 }
```

```
Enter a year: 2008 Enter
2008 is a leap year? true

Enter a year: 2002 Enter
2002 is a leap year? false
```

3.14 Problem: Lottery

Suppose you want to develop a program to play lottery. The program randomly generates a lottery of a 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 in exact order, the award is $10,000.
2. If all the digits in the user input match all the digits in the lottery, the award is $3,000.
3. If one digit in the user input matches a digit in the lottery, the award is $1,000.

The complete program is shown in Listing 3.9.

LISTING 3.9 Lottery.java

```
1 import java.util.Scanner;
2
3 public class Lottery {
4   public static void main(String[] args) {
5     // Generate a lottery
6     int lottery = (int)(Math.random() * 100);                      generate a lottery
7
8     // Prompt the user to enter a guess
9     Scanner input = new Scanner(System.in);
10    System.out.print("Enter your lottery pick (two digits): ");
11    int guess = input.nextInt();                                   enter a guess
12
13    //   Get digits from lottery
14    int  lotteryDigit1 = lottery / 10;
```

```
15     int lotteryDigit2 = lottery % 10;
16
17     // Get digits from guess
18     int guessDigit1 = guess / 10;
19     int guessDigit2 = guess % 10;
20
21     System.out.println("The lottery number is " + lottery);
22
23     // Check the guess
24     if (guess == lottery)
25       System.out.println("Exact match: you win $10,000");
26     else if (guessDigit2 == lotteryDigit1
27           && guessDigit1 == lotteryDigit2)
28       System.out.println("Match all digits: you win $3,000");
29     else if (guessDigit1 == lotteryDigit1
30           || guessDigit1 == lotteryDigit2
31           || guessDigit2 == lotteryDigit1
32           || guessDigit2 == lotteryDigit2)
33       System.out.println("Match one digit: you win $1,000");
34     else
35       System.out.println("Sorry, no match");
36   }
37 }
```

exact match?

match all digits?

match one digit?

```
Enter your lottery pick (two digits): 45 Enter
The lottery number is 12
Sorry, no match
```

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

line# / variable	6	11	14	15	18	19	33
lottery	34						
guess		23					
lotteryDigit1			3				
lotteryDigit2				4			
guessDigit1					2		
guessDigit2						3	
output							Match one digit: you win $1,000

The program generates a lottery using the `random()` method (line 6) and prompts the user to enter a guess (line 11). Note that `guess % 10` obtains the last digit from `guess` and `guess / 10` obtains the first digit from `guess`, since `guess` is a two-digit number (lines 18–19).

The program checks the guess against the lottery number in this order:

1. First check whether the guess matches the lottery exactly (line 24).
2. If not, check whether the reversal of the guess matches the lottery (lines 26–27).

3. If not, check whether one digit is in the lottery (lines 29–32).

4. If not, nothing matches.

3.15 switch Statements

The `if` statement in Listing 3.6, ComputeTax.java, makes selections based on a single `true` or `false` condition. There are four cases for computing taxes, which depend on the value of `status`. To fully account for all the cases, nested `if` statements were used. Overuse of nested `if` statements makes a program difficult to read. Java provides a `switch` statement to handle multiple conditions efficiently. You could write the following `switch` statement to replace the nested `if` statement in Listing 3.6:

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married filing jointly;
           break;
  case 2:  compute taxes for married filing separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(0);
}
```

The flow chart of the preceding `switch` statement is shown in Figure 3.5.

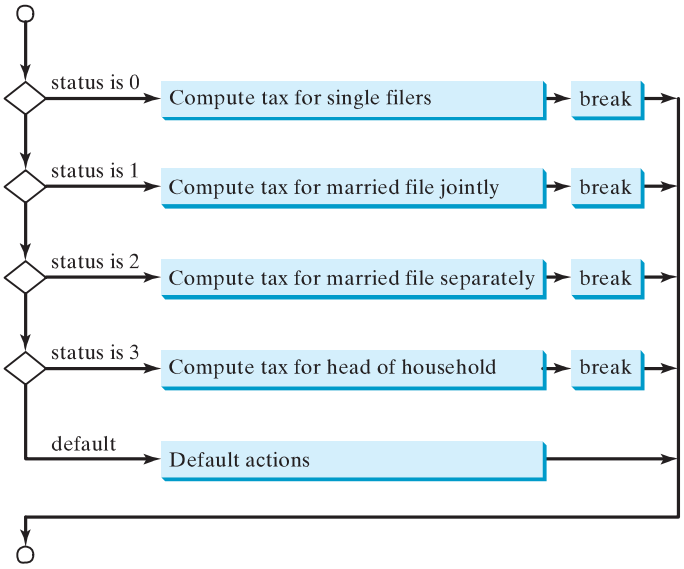


FIGURE 3.5 The `switch` statement checks all cases and executes the statements in the matched case.

This statement checks to see whether the status matches the value `0`, `1`, `2`, or `3`, in that order. If matched, the corresponding tax is computed; if not matched, a message is displayed. Here is the full syntax for the `switch` statement:

`switch` statement

```
switch (switch-expression) {
  case value1: statement(s)1;
               break;
```

```
    case value2: statement(s)2;
                 break;
    ...
    case valueN: statement(s)N;
                 break;
    default:     statement(s)-for-default;
}
```

The `switch` statement observes the following rules:

- The `switch-expression` must yield a value of `char`, `byte`, `short`, or `int` type and must always be enclosed in parentheses.
- The `value1`, ..., and `valueN` must have the same data type as the value of the `switch-expression`. Note that `value1`, ..., and `valueN` are constant expressions, meaning that they cannot contain variables, such as `1 + x`.
- When the value in a `case` statement matches the value of the `switch-expression`, the statements *starting from this case* are executed until either a `break` statement or the end of the switch statement is reached.
- The keyword `break` is optional. The `break` statement immediately ends the `switch` statement.
- The `default` case, which is optional, can be used to perform actions when none of the specified cases matches the `switch-expression`.
- The `case` statements are checked in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.

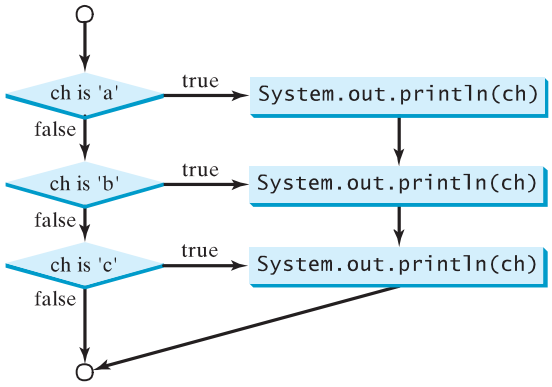
without **break**

fall-through behavior

Caution

Do not forget to use a `break` statement when one is needed. Once a case is matched, the statements starting from the matched case are executed until a `break` statement or the end of the `switch` statement is reached. This is referred to as *fall-through* behavior. For example, the following code prints character `a` three times if `ch` is `'a'`:

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```



Tip

To avoid programming errors and improve code maintainability, it is a good idea to put a comment in a case clause if `break` is purposely omitted.

3.16 Conditional Expressions

You might want to assign a value to a variable that is restricted by certain conditions. For example, the following statement assigns `1` to `y` if `x` is greater than `0`, and `-1` to `y` if `x` is less than or equal to `0`.

```
if (x > 0)
  y = 1;
else
  y = -1;
```

Alternatively, as in this example, you can use a conditional expression to achieve the same result.

```
y = (x > 0) ? 1 : -1;
```

Conditional expressions are in a completely different style, with no explicit `if` in the statement. The syntax is shown below:

```
boolean-expression ? expression1 : expression2;
```

conditional expression

The result of this conditional expression is `expression1` if `boolean-expression` is true; otherwise the result is `expression2`.

Suppose you want to assign the larger number between variable `num1` and `num2` to `max`. You can simply write a statement using the conditional expression:

```
max = (num1 > num2) ? num1 : num2;
```

For another example, the following statement displays the message "num is even" if `num` is even, and otherwise displays "num is odd."

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

Note

The symbols `?` and `:` appear together in a conditional expression. They form a conditional operator. It is called a *ternary operator* because it uses three operands. It is the only ternary operator in Java.

3.17 Formatting Console Output

If you wish to display only two digits after the decimal point in a floating-point value, you may write the code like this:

```
double x = 2.0 / 3;
System.out.println("x is " + (int)(x * 100) / 100.0);
```

```
x is 0.66
```

However, a better way to accomplish this task is to format the output using the `printf` method. The syntax to invoke this method is

printf

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

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

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 specifier consists of a percent sign (%) followed by a conversion code. Table 3.8 lists some frequently used simple specifiers:

TABLE 3.8 Frequently Used Specifiers

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);
```

items

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

Items must match the specifiers in order, in number, and in exact type. For example, the 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 specifier, as shown in the examples in Table 3.9.

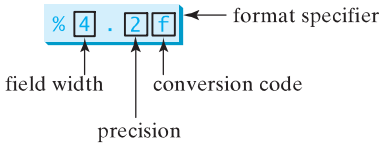
TABLE 3.9 Examples of Specifying Width and Precision

Example	*Output*
%5c	Output the character and add four spaces before the character item.
%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 less than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

The code presented in the beginning of this section for displaying only two digits after the decimal point in a floating-point value can be revised using the **printf** method as follows:

```
double x = 2.0 / 3;
System.out.printf("x is %4.2f", x);
```

```
display            x is 0.67
```

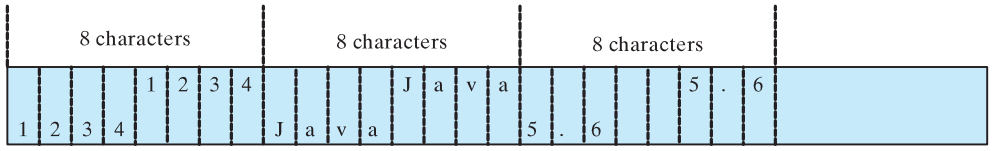


By default, the output is right justified. You can put the minus sign (-) in the specifier to specify that the item is left justified in the output within the specified field. For example, the following statements

left justify

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.6);
System.out.printf("%-8d%-8s%-8.1f \n", 1234, "Java", 5.6);
```

display



Caution

The items must match the specifiers in exact type. The item for the 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 specifier. To output a literal **%** in the format string, use **%%**.

3.18 Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated. Suppose that you have this expression:

```
3 + 4 * 4 > 5 * (4 + 3) - 1
```

What is its value? What is the execution order of the operators?

Arithmetically, the expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

The precedence rule defines precedence for operators, as shown in Table 3.10, which contains the operators you have learned so far. Operators are listed in decreasing order of precedence from top to bottom. Operators with the same precedence appear in the same group. (See Appendix C, "Operator Precedence Chart," for a complete list of Java operators and their precedence.)

precedence

If operators with the same precedence are next to each other, their *associativity* determines the order of evaluation. All binary operators except assignment operators are *left associative*. For example, since + and - are of the same precedence and are left associative, the expression

associativity

```
a - b + c - d  ---- equivalent ----  ((a - b) + c) - d
```

TABLE 3.10 Operator Precedence Chart

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Comparison)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

Assignment operators are *right associative*. Therefore, the expression

```
a = b += c = 5   ═══ equivalent ═══   a = (b += (c = 5))
```

Suppose `a`, `b`, and `c` are 1 before the assignment; after the whole expression is evaluated, `a` becomes 6, `b` becomes 6, and `c` becomes 5. Note that left associativity for the assignment operator would not make sense.

Note

Java has its own way to evaluate an expression internally. The result of a Java evaluation is the same as that of its corresponding arithmetic evaluation. Interested readers may refer to Supplement III.B for more discussions on how an expression is evaluated in Java *behind the scenes*.

behind the scenes

3.19 (GUI) Confirmation Dialogs

You have used `showMessageDialog` to display a message dialog box and `showInputDialog` to display an input dialog box. Occasionally it is useful to answer a question with a confirmation dialog box. A confirmation dialog can be created using the following statement:

```
int option =
  JOptionPane.showConfirmDialog
    (null, "Continue");
```

When a button is clicked, the method returns an option value. The value is `JOptionPane.YES_OPTION` (`0`) for the *Yes* button, `JOptionPane.NO_OPTION` (`1`) for the *No* button, and `JOptionPane.CANCEL_OPTION` (`2`) for the *Cancel* button.

You may rewrite the guess-birthday program in Listing 3.3 using confirmation dialog boxes, as shown in Listing 3.10. Figure 3.6 shows a sample run of the program for the day `19`.

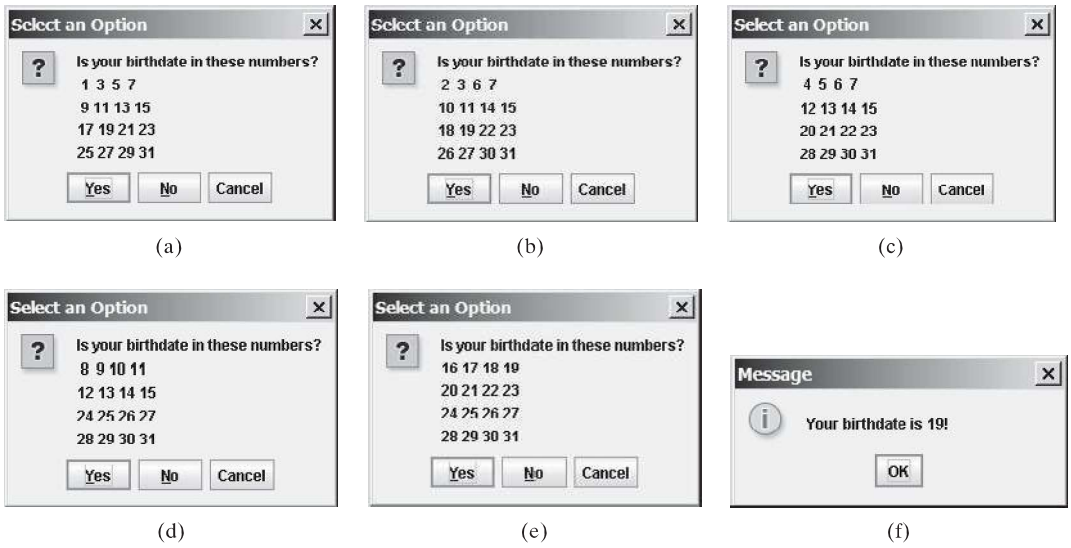


FIGURE 3.6 Click Yes in (a), Yes in (b), No in (c), No in (d), and Yes in (e).

LISTING 3.10 GuessBirthdayUsingConfirmationDialog.java

```
1  import javax.swing.JOptionPane;                                    import class
2
3  public class GuessBirthdayUsingConfirmationDialog {
4    public static void main(String[] args) {
5      String set1 =                                                  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 =                                                  set2
12       " 2  3  6  7\n" +
13       "10 11 14 15\n" +
14       "18 19 22 23\n" +
15       "26 27 30 31";
16
17     String set3 =                                                  set3
18       " 4  5  6  7\n" +
19       "12 13 14 15\n" +
20       "20 21 22 23\n" +
21       "28 29 30 31";
22
23     String set4 =                                                  set4
24       " 8  9 10 11\n" +
25       "12 13 14 15\n" +
26       "24 25 26 27\n" +
27       "28 29 30 31";
28
29     String set5 =                                                  set5
30       "16 17 18 19\n" +
31       "20 21 22 23\n" +
32       "24 25 26 27\n" +
33       "28 29 30 31";
34
```

confirmation dialog

in set1?

in set2?

in set3?

in set4?

in set5?

```
35      int day = 0;
36
37      // Prompt the user to answer questions
38      int answer = JOptionPane.showConfirmDialog(null,
39        "Is your birthday in these numbers?\n" + set1);
40
41      if (answer == JOptionPane.YES_OPTION)
42        day += 1;
43
44      answer = JOptionPane.showConfirmDialog(null,
45        "Is your birthday in these numbers?\n" + set2);
46
47      if (answer == JOptionPane.YES_OPTION)
48        day += 2;
49
50      answer = JOptionPane.showConfirmDialog(null,
51        "Is your birthday in these numbers?\n" + set3);
52
53      if (answer == JOptionPane.YES_OPTION)
54        day += 4;
55
56      answer = JOptionPane.showConfirmDialog(null,
57        "Is your birthday in these numbers?\n" + set4);
58
59      if (answer == JOptionPane.YES_OPTION)
60        day += 8;
61
62      answer = JOptionPane.showConfirmDialog(null,
63        "Is your birthday in these numbers?\n" + set5);
64
65      if (answer == JOptionPane.YES_OPTION)
66        day += 16;
67
68      JOptionPane.showMessageDialog(null, "Your birthday is " +
69        day + "!");
70    }
71  }
```

The program displays confirmation dialog boxes to prompt the user to answer whether a number is in Set1 (line 38), Set2 (line 44), Set3 (line 50), Set4 (line 56), and Set5 (line 62). If the answer is Yes, the first number in the set is added to **day** (lines 42, 48, 54, 60, and 66).

Key Terms

Boolean expression 72
Boolean value 72
boolean type 72
break statement 94
conditional operator 90
dangling-**else** ambiguity 82
fall-through behavior 94
operator associativity 97
operator precedence 97
selection statement 74
short-circuit evaluation 90

Chapter Summary

1. A **boolean** variable stores a **true** or **false** value.

2. The relational operators (<, <=, ==, !=, >, >=) work with numbers and characters, and yield a Boolean value.

3. The Boolean operators `&&`, `||`, `!`, and `^` operate with Boolean values and variables.

4. When evaluating `p1 && p2`, Java first evaluates `p1` and then evaluates `p2` if `p1` is `true`; if `p1` is `false`, it does not evaluate `p2`. When evaluating `p1 || p2`, Java first evaluates `p1` and then evaluates `p2` if `p1` is `false`; if `p1` is `true`, it does not evaluate `p2`. Therefore, `&&` is referred to as the conditional or short-circuit AND operator, and `||` is referred to as the conditional or short-circuit OR operator.

5. Selection statements are used for programming with alternative courses. There are several types of selection statements: `if` statements, `if ... else` statements, nested `if` statements, `switch` statements, and conditional expressions.

6. The various `if` statements all make control decisions based on a Boolean expression. Based on the `true` or `false` evaluation of the expression, these statements take one of two possible courses.

7. The `switch` statement makes control decisions based on a switch expression of type `char`, `byte`, `short`, or `int`.

8. The keyword `break` is optional in a switch statement, but it is normally used at the end of each case in order to terminate the remainder of the `switch` statement. If the `break` statement is not present, the next `case` statement will be executed.

Review Questions

Section 3.2

3.1 List six comparison operators.

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

```
boolean b = true;
i = (int)b;
```

```
int i = 1;
boolean b = (boolean)i;
```

Sections 3.3–3.11

3.3 What is the printout of the code in (a) and (b) if `number` is `30` and `35`, respectively?

```
if (number % 2 == 0)
  System.out.println(number + " is even.");

  System.out.println(number + " is odd.");
```

(a)

```
if (number % 2 == 0)
  System.out.println(number + " is even.");
else
  System.out.println(number + " is odd.");
```

(b)

3.4 Suppose `x = 3` and `y = 2`; show the output, if any, of the following code. What is the output if `x = 3` and `y = 4`? What is the output if `x = 2` and `y = 2`? Draw a flow chart of the code:

```
if (x > 2) {
  if (y > 2) {
```

```
    z = x + y;
    System.out.println("z is " + z);
  }
}
else
  System.out.println("x is " + x);
```

3.5 Which of the following statements are equivalent? Which ones are correctly indented?

```
if (i > 0) if
(j > 0)
x = 0; else
if (k > 0) y = 0;
else z = 0;
```

(a)

```
if (i > 0) {
  if (j > 0)
    x = 0;
  else if (k > 0)
    y = 0;
}
else
  z = 0;
```

(b)

```
if (i > 0)
  if (j > 0)
    x = 0;
  else if (k > 0)
    y = 0;
  else
    z = 0;
```

(c)

```
if (i > 0)
  if (j > 0)
    x = 0;
  else if (k > 0)
    y = 0;
else
  z = 0;
```

(d)

3.6 Suppose `x = 2` and `y = 3`. Show the output, if any, of the following code. What is the output if `x = 3` and `y = 2`? What is the output if `x = 3` and `y = 3`?

(*Hint*: Indent the statement correctly first.)

```
if (x > 2)
  if (y > 2) {
    int z = x + y;
    System.out.println("z is " + z);
  }
else
  System.out.println("x is " + x);
```

3.7 Are the following two statements equivalent?

```
if (income <= 10000)
  tax = income * 0.1;
else if (income <= 20000)
  tax = 1000 +
    (income - 10000) * 0.15;
```

```
if (income <= 10000)
  tax = income * 0.1;
else if (income > 10000 &&
         income <= 20000)
  tax = 1000 +
    (income - 10000) * 0.15;
```

3.8 Which of the following is a possible output from invoking `Math.random()`?

`323.4`, `0.5`, `34`, `1.0`, `0.0`, `0.234`

3.9 How do you generate a random integer `i` such that $0 \le i < 20$? How do you generate a random integer `i` such that $10 \le i < 20$? How do you generate a random integer `i` such that $10 \le i \le 50$?

3.10 Write an `if` statement that assigns `1` to `x` if `y` is greater than `0`.

3.11 (a) Write an `if` statement that increases `pay` by 3% if `score` is greater than `90`. (b) Write an `if` statement that increases `pay` by 3% if `score` is greater than `90`, otherwise increases `pay` by 1%.

3.12 What is wrong in the following code?

```
if (score >= 60.0)
  grade = 'D';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 90.0)
  grade = 'A';
else
  grade = 'F';
```

3.13 Rewrite the following statement using a Boolean expression:

```
if (count % 10 == 0)
  newLine = true;
else
  newLine = false;
```

Sections 3.12–3.14

3.14 Assuming that `x` is `1`, show the result of the following Boolean expressions.

```
(true) && (3 > 4)
!(x > 0) && (x > 0)
(x > 0) || (x < 0)
(x != 0) || (x == 0)
(x >= 0) || (x < 0)
(x != 1) == !(x == 1)
```

3.15 Write a Boolean expression that evaluates to `true` if a number stored in variable `num` is between `1` and `100`.

3.16 Write a Boolean expression that evaluates to `true` if a number stored in variable `num` is between `1` and `100` or the number is negative.

3.17 Assume that `x` and `y` are `int` type. Which of the following are legal Java expressions?

```
x > y > 0
x = y && y
x /= y
x or y
x and y
(x != 0) || (x = 0)
```

3.18 Suppose that `x` is `1`. What is `x` after the evaluation of the following expression?

```
(x >= 1) && (x++ > 1)
(x > 1) && (x++ > 1)
```

3.19 What is the value of the expression `ch >= 'A' && ch <= 'Z'` if `ch` is `'A'`, `'p'`, `'E'`, or `'5'`?

3.20 Suppose, when you run the program, you enter input `2 3 6` from the console. What is the output?

```
public class Test {
  public static void main(String[] args) {
    java.util.Scanner input = new java.util.Scanner(System.in);
    double x = input.nextDouble();
```

```
    double y = input.nextDouble();
    double z = input.nextDouble();

    System.out.println("(x < y && y < z) is " + (x < y && y < z));
    System.out.println("(x < y || y < z) is " + (x < y || y < z));
    System.out.println("!(x < y) is " + !(x < y));
    System.out.println("(x + y < z) is " + (x + y < z));
    System.out.println("(x + y < z) is " + (x + y < z));
  }
}
```

3.21 Write a Boolean expression that evaluates `true` if `age` is greater than `13` and less than `18`.

3.22 Write a Boolean expression that evaluates `true` if `weight` is greater than `50` or height is greater than `160`.

3.23 Write a Boolean expression that evaluates `true` if `weight` is greater than `50` and height is greater than `160`.

3.24 Write a Boolean expression that evaluates `true` if either `weight` is greater than `50` or height is greater than `160`, but not both.

Section 3.15

3.25 What data types are required for a `switch` variable? If the keyword `break` is not used after a case is processed, what is the next statement to be executed? Can you convert a `switch` statement to an equivalent `if` statement, or vice versa? What are the advantages of using a `switch` statement?

3.26 What is `y` after the following `switch` statement is executed?

```
x = 3; y = 3;
switch (x + 3) {
  case 6:  y = 1;
  default: y += 1;
}
```

3.27 Use a `switch` statement to rewrite the following `if` statement and draw the flow chart for the `switch` statement:

```
if (a == 1)
  x += 5;
else if (a == 2)
  x += 10;
else if (a == 3)
  x += 16;
else if (a == 4)
  x += 34;
```

3.28 Write a `switch` statement that assigns a `String` variable `dayName` with Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, if `day` is `0`, `1`, `2`, `3`, `4`, `5`, `6`, accordingly.

Section 3.16

3.29 Rewrite the following `if` statement using the conditional operator:

```
if (count % 10 == 0)
  System.out.print(count + "\n");
else
  System.out.print(count + " ");
```

3.30 Rewrite the following statement using a conditional expression:

```
if (temperature > 90)
  pay = pay * 1.5;
else
  pay = pay * 1.1;
```

Section 3.17

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

3.32 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);
```

3.33 Show the output of the following statements.

```
(a) System.out.printf("amount is %f %e\n", 32.32, 32.32);
(b) System.out.printf("amount is %5.4f %5.4e\n", 32.32, 32.32);
(c) System.out.printf("%6b\n", (1 > 2));
(d) System.out.printf("%6s\n", "Java");
(e) System.out.printf("%-6b%s\n", (1 > 2), "Java");
(f) System.out.printf("%6b%-s\n", (1 > 2), "Java");
```

3.34 How do you create a formatted string?

Section 3.18

3.35 List the precedence order of the Boolean operators. Evaluate the following expressions:

```
true || true && false
true && true || false
```

3.36 True or false? All the binary operators except `=` are left associative.

3.37 Evaluate the following expressions:

```
2 * 2 - 3 > 2 && 4 - 2 > 5
2 * 2 - 3 > 2 || 4 - 2 > 5
```

3.38 Is `(x > 0 && x < 10)` the same as `((x > 0) && (x < 10))`? Is `(x > 0 || x < 10)` the same as `((x > 0) || (x < 10))`? Is `(x > 0 || x < 10 && y < 0)` the same as `(x > 0 || (x < 10 && y < 0))`?

Section 3.19

3.39 How do you display a confirmation dialog? What value is returned when invoking `JOptionPane.showConfirmDialog`?

Programming Exercises

Pedagogical Note

For each exercise, students should carefully analyze the problem requirements and design strategies for solving the problem before coding.

think before coding

document analysis and design

Pedagogical Note

Instructors may ask students to document analysis and design for selected exercises. Students should use their own words to analyze the problem, including the input, output, and what needs to be computed, and describe how to solve the problem in pseudocode.

learn from mistakes

Debugging Tip

Before you ask for help, read and explain the program to yourself, and trace it using several representative inputs by hand or using an IDE debugger. You learn how to program by debugging your own mistakes.

Section 3.2

3.1* (*Algebra: solving quadratic equations*) The two roots of a quadratic equation $ax^2 + bx + c = 0$ can be obtained using the following formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ and } r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$b^2 - 4ac$ is called the discriminant of the quadratic equation. If it is positive, the equation has two real roots. If it is zero, the equation has one root. If it is negative, the equation has no real roots.

Write a program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, display two roots. If the discriminant is 0, display one root. Otherwise, display "The equation has no real roots".

Note you can use `Math.pow(x, 0.5)` to compute $\sqrt{x}$. Here are some sample runs.

```
Enter a, b, c: 1.0 3 1 Enter
The roots are -0.381966 and -2.61803
```

```
Enter a, b, c: 1 2.0 1 Enter
The root is -1
```

```
Enter a, b, c: 1 2 3 Enter
The equation has no real roots
```

3.2 (*Checking whether a number is even*) Write a program that reads an integer and checks whether it is even. Here are the sample runs of this program:

```
Enter an integer: 25 Enter
Is 25 an even number? false
```

```
Enter an integer: 2000 Enter
Is 2000 an even number? true
```

Sections 3.3–3.8

3.3* (*Algebra: solving* 2×2 *linear equations*) You can use Cramer's rule to solve the following 2×2 system of linear equation:

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \qquad x = \frac{ed - bf}{ad - bc} \qquad y = \frac{af - ec}{ad - bc}$$

Write a program that prompts the user to enter **a**, **b**, **c**, **d**, **e**, and **f** and display the result. If $ad - bc$ is **0**, report that "The equation has no solution".

```
Enter a, b, c, d, e, f: 9.0 4.0 3.0 -5.0 -6.0 -21.0 Enter
x is -2.0 and y is 3.0
```

```
Enter a, b, c, d, e, f: 1.0 2.0 2.0 4.0 4.0 5.0 Enter
The equation has no solution
```

3.4** (*Game: learning addition*) Write a program that generates two integers under 100 and prompts the user to enter the sum of these two integers. The program then reports true if the answer is correct, false otherwise. The program is similar to Listing 3.1.

3.5** (*Game: addition for three numbers*) The program in Listing 3.1 generates two integers and prompts the user to enter the sum of these two integers. Revise the program to generate three single-digit integers and prompt the user to enter the sum of these three integers.

3.6* (*Health application: BMI*) Revise Listing 3.5, ComputeBMI.java, to let the user enter weight, feet, and inches. For example, if a person is **5** feet and **10** inches, you will enter **5** for feet and **10** for inches.

3.7 (*Financial application: monetary units*) Modify Listing 2.10, ComputeChange.java, to display the nonzero denominations only, using singular words for single units such as **1** dollar and **1** penny, and plural words for more than one unit such as **2** dollars and **3** pennies. (Use input **23.67** to test your program.)

Video Note
Sort three integers

3.8* (*Sorting three integers*) Write a program that sorts three integers. The integers are entered from the input dialogs and stored in variables **num1**, **num2**, and **num3**, respectively. The program sorts the numbers so that $num1 \le num2 \le num3$.

3.9 (*Business: checking ISBN*) An **ISBN** (International Standard Book Number) consists of 10 digits $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}$. The last digit d_{10} is a checksum, which is calculated from the other nine digits using the following formula:

$$(d_1 \times 1 + d_2 \times 2 + d_3 \times 3 + d_4 \times 4 + d_5 \times 5 + d_6 \times 6 + d_7 \times 7 + d_8 \times 8 + d_9 \times 9) \% 11$$

If the checksum is **10**, the last digit is denoted X according to the ISBN convention. Write a program that prompts the user to enter the first 9 digits and displays the 10-digit ISBN (including leading zeros). Your program should read the input as an integer. For example, if you enter 013601267, the program should display 0136012671.

3.10* (*Game: addition quiz*) Listing 3.4, SubtractionQuiz.java, randomly generates a subtraction question. Revise the program to randomly generate an addition question with two integers less than **100**.

Sections 3.9–3.19

3.11* (*Finding the number of days in a month*) Write a program that prompts the user to enter the month and year and displays the number of days in the month. For example, if the user entered month 2 and year 2000, the program should display that February 2000 has 29 days. If the user entered month 3 and year 2005, the program should display that March 2005 has 31 days.

3.12 (*Checking a number*) Write a program that prompts the user to enter an integer and checks whether the number is divisible by both 5 and 6, or neither of them, or just one of them. Here are some sample runs for inputs 10, 30, and 23.

```
10 is divisible by 5 or 6, but not both
30 is divisible by both 5 and 6
23 is not divisible by either 5 or 6
```

3.13 (*Financial application: computing taxes*) Listing 3.6, ComputeTax.java, gives the source code to compute taxes for single filers. Complete Listing 3.6 to give the complete source code.

3.14 (*Game: head or tail*) Write a program that lets the user guess the head or tail of a coin. The program randomly generates an integer 0 or 1, which represents head or tail. The program prompts the user to enter a guess and reports whether the guess is correct or incorrect.

3.15* (*Game: lottery*) Revise Listing 3.9, Lottery.java, to generate a lottery of a three-digit number. The program prompts the user to enter a three-digit number and determines whether the user wins according to the following rule:

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

3.16 (*Random character*) Write a program that displays a random uppercase letter using the `Math.random()` method.

3.17* (*Game: scissor, rock, paper*) Write a program that plays the popular scissor-rock-paper game. (A scissor can cut a paper, a rock can knock a scissor, and a paper can wrap a rock.) The program randomly generates a number 0, 1, or 2 representing scissor, rock, and paper. The program prompts the user to enter a number 0, 1, or 2 and displays a message indicating whether the user or the computer wins, loses, or draws. Here are sample runs:

```
scissor (0), rock (1), paper (2): 1 Enter
The computer is scissor. You are rock. You won
```

```
scissor (0), rock (1), paper (2): 2 Enter
The computer is paper. You are paper too. It is a draw
```

3.18* (*Using the input dialog box*) Rewrite Listing 3.8, LeapYear.java, using the input dialog box.

3.19 (*Validating triangles*) Write a program that reads three edges for a triangle and determines whether the input is valid. The input is valid if the sum of any two edges is greater than the third edge. Here are the sample runs of this program:

```
Enter three edges: 1 2.5 1 Enter
Can edges 1, 2.5, and 1 form a triangle? false
```

```
Enter three edges: 2.5 2 1 Enter
Can edges 2.5, 2, and 1 form a triangle? true
```

3.20* (*Science: wind-chill temperature*) Exercise 2.17 gives a formula to compute the wind-chill temperature. The formula is valid for temperature in the range between −58°F and 41°F and wind speed greater than or equal to **2**. Write a program that prompts the user to enter a temperature and a wind speed. The program displays the wind-chill temperature if the input is valid, otherwise displays a message indicating whether the temperature and/or wind speed is invalid.

Comprehensives

3.21** (*Science: day of the week*) Zeller's congruence is an algorithm developed by Christian Zeller to calculate the day of the week. The formula is

$$h = \left(q + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + k + \left\lfloor \frac{k}{4} \right\rfloor + \left\lfloor \frac{j}{4} \right\rfloor + 5j\right) \% 7$$

where

- **h** is the day of the week (0: Saturday, 1: Sunday, 2: Monday, 3: Tuesday, 4: Wednesday, 5: Thursday, 6: Friday).
- **q** is the day of the month.
- **m** is the month (3: March, 4: April, ..., 12: December). January and February are counted as months 13 and 14 of the previous year.
- **j** is the century (i.e., $\left\lfloor \frac{year}{100} \right\rfloor$).
- **k** is the year of the century (i.e., $year \% 7$).

Write a program that prompts the user to enter a year, month, and day of the month, and displays the name of the day of the week. Here are some sample runs:

```
Enter year: (e.g., 2008): 2002 Enter
Enter month: 1-12: 3 Enter
Enter the day of the month: 1-31: 26 Enter
Day of the week is Tuesday
```

```
Enter year: (e.g., 2008): 2011 Enter
Enter month: 1-12: 5 Enter
Enter the day of the month: 1-31: 2 Enter
Day of the week is Thursday
```

(*Hint*: $\lfloor n \rfloor = (\text{int})n$ for a positive n. January and February are counted as **13** and **14** in the formula. So you need to convert the user input **1** to **13** and **2** to **14** for the month and change the year to the previous year.)

3.22** (*Geometry: point in a circle?*) Write a program that prompts the user to enter a point (`x`, `y`) and checks whether the point is within the circle centered at (`0`, `0`) with radius `10`. For example, (`4`, `5`) is inside the circle and (`9`, `9`) is outside the circle, as shown in Figure 3.7(a).

(*Hint*: A point is in the circle if its distance to (`0`, `0`) is less than or equal to `10`. The formula for computing the distance is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.) Two sample runs are shown below.)

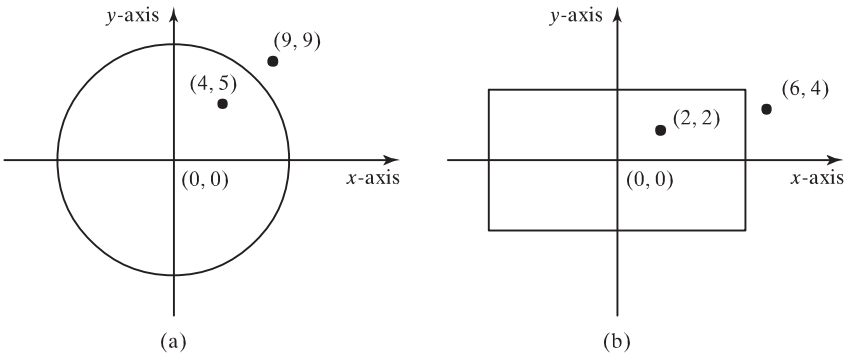


FIGURE 3.7 (a) Points inside and outside of the circle; (b) Points inside and outside of the rectangle.

```
Enter a point with two coordinates: 4 5 Enter
Point (4.0, 5.0) is in the circle
```

```
Enter a point with two coordinates: 9 9 Enter
Point (9.0, 9.0) is not in the circle
```

3.23** (*Geometry: point in a rectangle?*) Write a program that prompts the user to enter a point (`x, y`) and checks whether the point is within the rectangle centered at (`0`, `0`) with width `10` and height `5`. For example, (`2`, `2`) is inside the rectangle and (`6`, `4`) is outside the circle, as shown in Figure 3.7(b).

(*Hint*: A point is in the rectangle if its horizontal distance to (`0`, `0`) is less than or equal to `10 / 2` and its vertical distance to (`0`, `0`) is less than or equal to `5 / 2`.] Here are two sample runs. Two sample runs are shown below.)

```
Enter a point with two coordinates: 2 2 Enter
Point (2.0, 2.0) is in the rectangle
```

```
Enter a point with two coordinates: 6 4 Enter
Point (6.0, 4.0) is not in the rectangle
```

3.24** (*Game: picking a card*) Write a program that simulates picking a card from a deck of `52` cards. Your program should display the rank (`Ace`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `Jack`, `Queen`, `King`) and suit (`Clubs`, `Diamonds`, `Hearts`, `Spades`) of the card. Here is a sample run of the program:

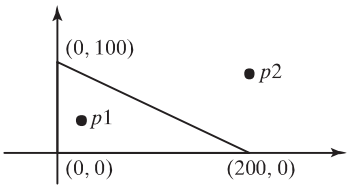
```
The card you picked is Jack of Hearts
```

3.25** (*Computing the perimeter of a triangle*) Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of any two edges is greater than the third edge.

3.26 (*Using the* `&&`, `||` *and* `^` *operators*) Write a program that prompts the user to enter an integer and determines whether it is divisible by 5 and 6, whether it is divisible by 5 or 6, and whether it is divisible by 5 or 6, but not both. Here is a sample run of this program:

```
Enter an integer: 10 Enter
Is 10 divisible by 5 and 6? false
Is 10 divisible by 5 or 6? true
Is 10 divisible by 5 or 6, but not both? true
```

3.27** (*Geometry: points in triangle?*) Suppose a right triangle is placed in a plane as shown below. The right-angle point is placed at (0, 0), and the other two points are placed at (200, 0), and (0, 100). Write a program that prompts the user to enter a point with x- and y-coordinates and determines whether the point is inside the triangle. Here are the sample runs:




```
Enter a point's x- and y-coordinates: 100.5 25.5 Enter
The point is in the triangle
```

```
Enter a point's x- and y-coordinates: 100.5 50.5 Enter
The point is not in the triangle
```

3.28** (*Geometry: two rectangles*) Write a program that prompts the user to enter the center *x*-, *y*-coordinates, width, and height of two rectangles and determines whether the second rectangle is inside the first or overlaps with the first, as shown in Figure 3.8.

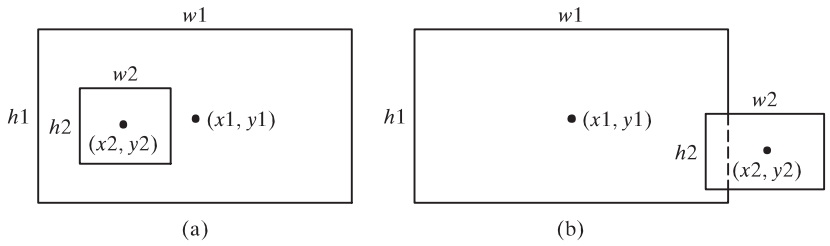


FIGURE 3.8 (a) A rectangle is inside another one. (b) A rectangle overlaps another one.

Here are the sample runs:

```
Enter r1's center x-, y-coordinates, width, and height:
  2.5 4 2.5 43  Enter
Enter r2's center x-, y-coordinates, width, and height:
  1.5 5 0.5 3  Enter
r2 is inside r1
```

```
Enter r1's center x-, y-coordinates, width, and height:
  1 2 3 5.5  Enter
Enter r2's center x-, y-coordinates, width, and height:
  3 4 4.5 5  Enter
r2 overlaps r1
```

```
Enter r1's center x-, y-coordinates, width, and height:
  1 2 3 3  Enter
Enter r2's center x-, y-coordinates, width, and height:
  40 45 3 2  Enter
r2 does not overlap r1
```

3.29** (*Geometry: two circles*) Write a program that prompts the user to enter the center coordinates and radii of two circles and determines whether the second circle is inside the first or overlaps with the first, as shown in Figure 3.9.

(*Hint*: circle2 is inside circle1 if the distance between the two centers <= `|r1 - r2|` and circle2 overlaps circle1 if the distance between the two centers <= `r1 + r2`.)

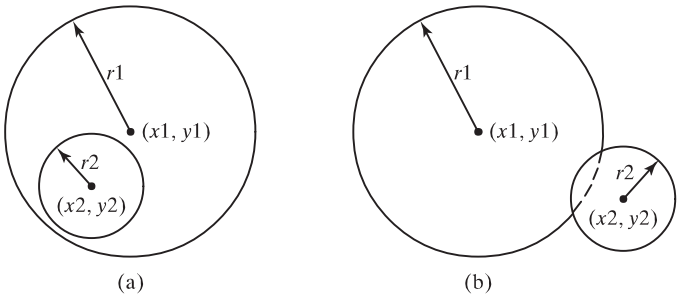


FIGURE 3.9 (a) A circle is inside another circle. (b) A circle overlaps another circle.

Here are the sample runs:

```
Enter circle1's center x-, y-coordinates, and radius:
  0.5 5.1 13  Enter
Enter circle2's center x-, y-coordinates, and radius:
  1 1.7 4.5  Enter
circle2 is inside circle1
```

```
Enter circle1's center x-, y-coordinates, and radius:
  3.4 5.7 5.5  Enter
Enter circle2's center x-, y-coordinates, and radius:
  6.7 3.5 3  Enter
circle2 overlaps circle1
```

```
Enter circle1's center x-, y-coordinates, and radius:
  3.4 5.5 1  Enter
Enter circle2's center x-, y-coordinates, and radius:
  5.5 7.2 1  Enter
circle2 does not overlap circle1
```