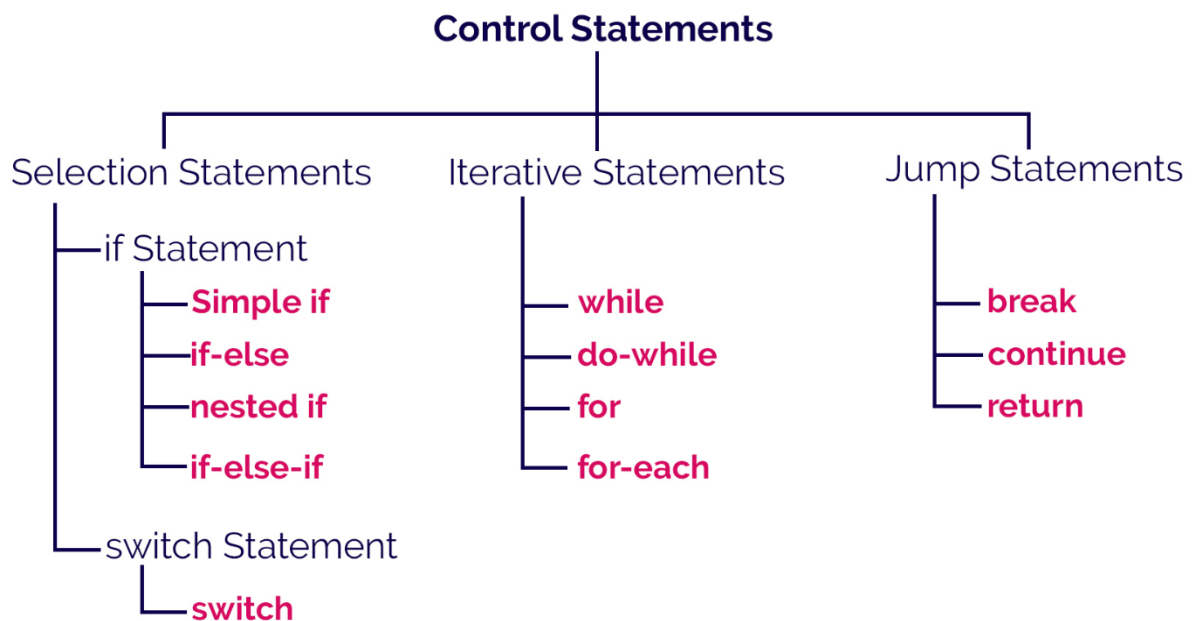


Chapter 3: Java Control Statements

In java, the default execution flow of a program is a sequential order. But the sequential order of execution flow may not be suitable for all situations. Sometimes, we may want to jump from line to another line, we may want to skip a part of the program, or sometimes we may want to execute a part of the program again and again. To solve this problem, java provides control statements.



1. Selection Statement

Like all high-level programming languages, Java provides *selection statements*: statements that let you choose actions with alternative courses. The program can decide which statements to execute based on a condition. Selection statements use conditions that are Boolean expressions. A *Boolean expression* is an expression that evaluates to a *Boolean value*: **true** or **false**.

2. boolean Data Type

The boolean data type declares a variable with the value either true or false.

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 relational operators (also known as comparison operators), shown in Table 3.1, which can be used to compare two values (assume radius is 5 in the table).

TABLE 3.1 Relational Operators

<i>Java Operator</i>	<i>Mathematics Symbol</i>	<i>Name</i>	<i>Example (radius is 5)</i>	<i>Result</i>
<	<	Less than	<code>radius < 0</code>	<code>false</code>
<=	≤	Less than or equal to	<code>radius <= 0</code>	<code>false</code>
>	>	Greater than	<code>radius > 0</code>	<code>true</code>
>=	≥	Greater than or equal to	<code>radius >= 0</code>	<code>true</code>
==	=	Equal to	<code>radius == 0</code>	<code>false</code>
!=	≠	Not equal to	<code>radius != 0</code>	<code>true</code>

3. if Statement

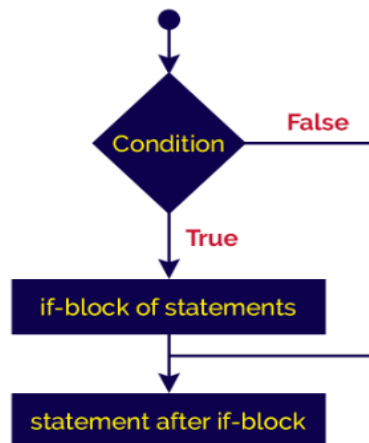
Java has several types of selection statements: one-way **if** statements, two-way **if-else** statements, nested **if** statements, multi-way **if-else** statements, **switch** statements, and **conditional operators**.

In java, we use the if statement to test a condition and decide the execution of a block of statements based on that condition result. The if statement checks, the given condition then decides the execution of a block of statements. If the condition is True, then the block of statements is executed and if it is False, then the block of statements is ignored. The syntax and execution flow of if the statement is as follows.

Let's look at the following example java code.

Syntax

```
if(condition){  
    if-block of statements;  
    ...  
}  
statement after if-block;  
...
```

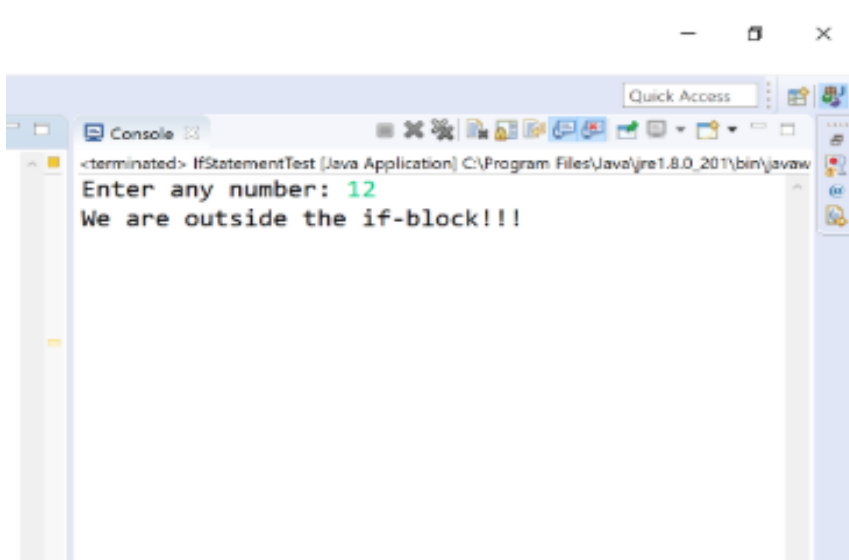
Flow of execution

Let's look at the following example java code.

Java Program

```
import java.util.Scanner;  
  
public class IfStatementTest {  
  
    public static void main(String[] args) {  
  
        Scanner read = new Scanner(System.in);  
        System.out.print("Enter any number: ");  
        int num = read.nextInt();  
  
        if((num % 5) == 0) {  
            System.out.println("We are inside the if-block!");  
            System.out.println("Given number is divisible by 5!!");  
        }  
  
        System.out.println("We are outside the if-block!!!");  
  
    }  
  
}
```

When we run this code, it produces the following output.



```
<terminated> IfStatementTest [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw
Enter any number: 12
We are outside the if-block!!!
```

In the above execution, the number 12 is not divisible by 5. So, the condition becomes False and the condition is evaluated as False. Then the if statement ignores the execution of its block of statements.

When we enter a number, which is divisible by 5, then it produces the output as follows.



```
<terminated> IfStatementTest [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw
Enter any number: 20
We are inside the if-block!
Given number is divisible by 5!!
We are outside the if-block!!!
```

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 as follows:

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

The flowchart in Figure 3.1a illustrates how Java executes the syntax of an **if** statement.

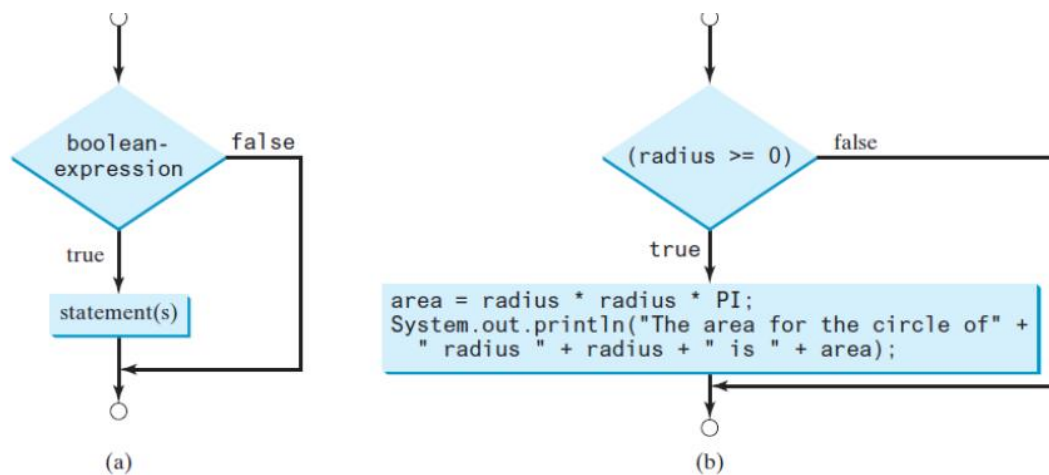


FIGURE 3.1 An **if** statement executes statements if the **boolean-expression** evaluates to **true**.

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 flowchart of the preceding statement is shown in Figure 3.1b. 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) 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**, the program displays **HiFive**. If the number is divisible by **2**, it displays **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.print("Enter an integer: ");
7         int number = input.nextInt();
8
9         if (number % 5 == 0)
10            System.out.println("HiFive");
11
12        if (number % 2 == 0)
13            System.out.println("HiEven");
14    }
15 }
```

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

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

The program prompts the user to enter an integer (lines 6–7) and displays **HiFive** if it is divisible by **5** (lines 9–10) and **HiEven** if it is divisible by **2** (lines 12–13).

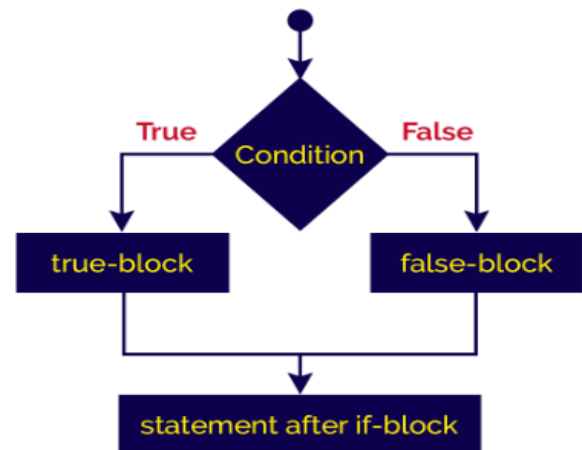
3.1 Two-Way if-else Statements

An **if-else** statement decides the execution path based on whether the condition is true or false. A one-way **if** statement performs 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-else**

Syntax

```
if(condition){  
    true-block of statements;  
    ...  
}  
else{  
    false-block of statements;  
    ...  
}  
statement after if-block;  
...
```


Flow of execution



Let's look at the following example java code.

```
import java.util.Scanner;  
  
public class IfElseStatementTest {  
    public static void main(String[] args) {  
        Scanner read = new Scanner(System.in);  
        System.out.print("Enter any number: ");  
        int num = read.nextInt();  
  
        if((num % 2) == 0) {  
            System.out.println("We are inside the true-block!");  
            System.out.println("Given number is EVEN number!!");  
        }  
        else {  
            System.out.println("We are inside the false-block!");  
            System.out.println("Given number is ODD number!!");  
        }  
  
        System.out.println("We are outside the if-block!!!");  
    }  
}
```

When we run this code, it produces the following output.



```
<terminated> IfElseStatementTest [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\ja
Enter any number: 16
We are inside the true-block!
Given number is EVEN number!!
We are outside the if-block!!!
```

3.2 Nested if and Multi-Way if-else Statements

An **if** statement can be inside another **if** statement to form a nested **if** statement. 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:

```
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 3.3a, for instance, prints a letter grade according to the score, with multiple alternatives.

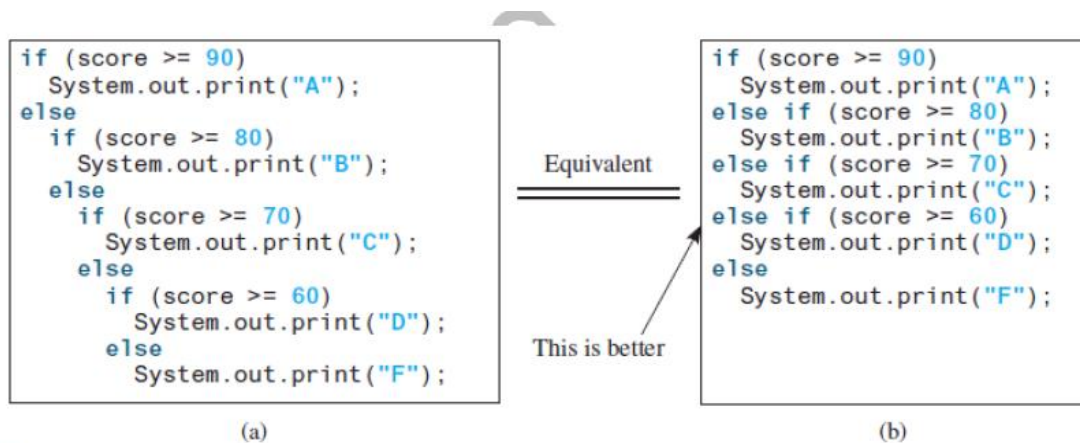


FIGURE 3.3 A preferred format for multiple alternatives is shown in (b) using a multi-way `if-else` statement.

The **if** statement in Figure 3.3a is equivalent to the **if** statement in Figure 3.3b. In fact, Figure 3.3b is the preferred coding style for multiple alternative **if** statements. This style, called *multi-way if-else statements*, avoids deep indentation and makes the program easy to read.

The execution of this **if** statement proceeds as shown in Figure 3.4. The first condition (**score >= 90**) is tested. If it is **true**, the grade is **A**. If it is **false**, the second condition (**score >= 80**) is tested. If the second condition is **true**, the grade is **B**. If that condition is **false**, the third condition and the rest of the conditions (if necessary) are tested until a condition is met or all of the conditions prove to be **false**. If all of the conditions are **false**, the grade is **F**. Note a condition is tested only when all of the conditions that come before it are **false**.

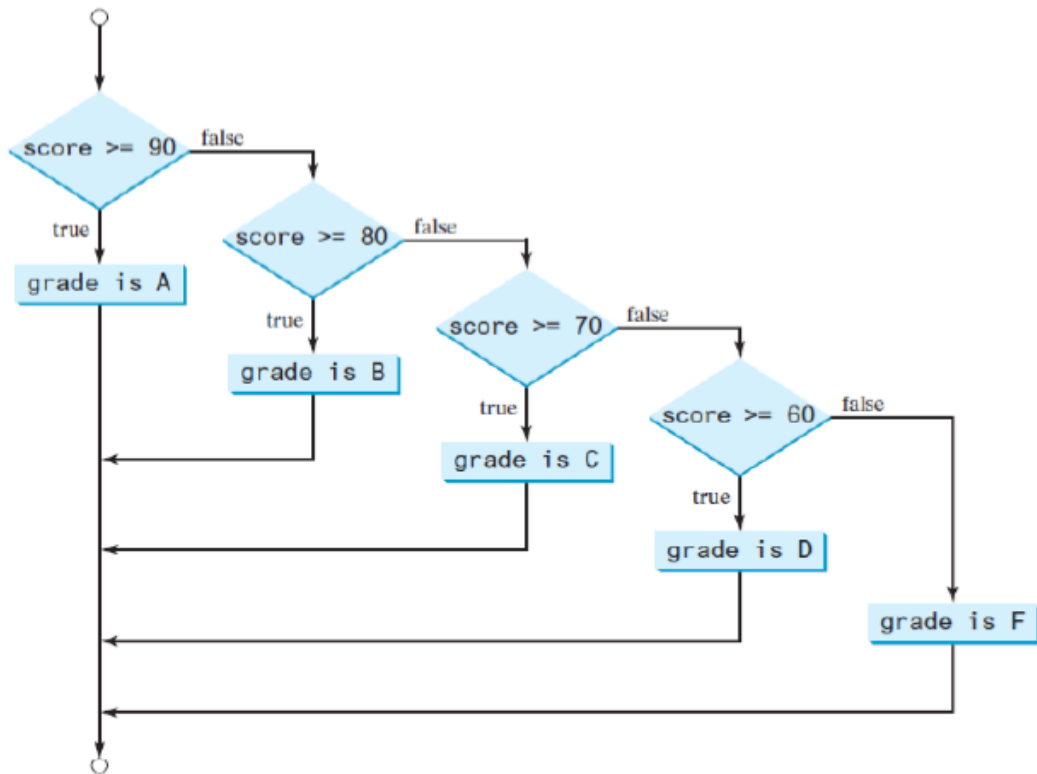


FIGURE 3.4 You can use a multi-way `if-else` statement to assign a grade.

4. 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 to form a compound Boolean expression. *Logical operators*, also known as *Boolean operators*, operate on Boolean values to create a new Boolean value.

Listing 3.6 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 3.6 TestBooleanOperators.java

```
1 import java.util.Scanner;
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();
11
12        if (number % 2 == 0 && number % 3 == 0)
13            System.out.println(number + " is divisible by 2 and 3.");
14
15        if (number % 2 == 0 || number % 3 == 0)
16            System.out.println(number + " is divisible by 2 or 3.");
17
18        if (number % 2 == 0 ^ number % 3 == 0)
19            System.out.println(number +
20                " is divisible by 2 or 3, but not both.");
21    }
22 }
```

```
Enter an integer: 4  Enter
4 is divisible by 2 or 3.
4 is divisible by 2 or 3, but not both.
```

```
Enter an integer: 18  Enter
18 is divisible by 2 and 3.
18 is divisible by 2 or 3.
```

(number % 2 == 0 && number % 3 == 0) (line 12) checks whether the number is divisible by both 2 and 3. **(number % 2 == 0 || number % 3 == 0)** (line 15) checks whether the number is divisible by 2 or by 3. **(number % 2 == 0 ^ number % 3 == 0)** (line 18) checks whether the number is divisible by 2 or 3, but not both.

5. switch Statements

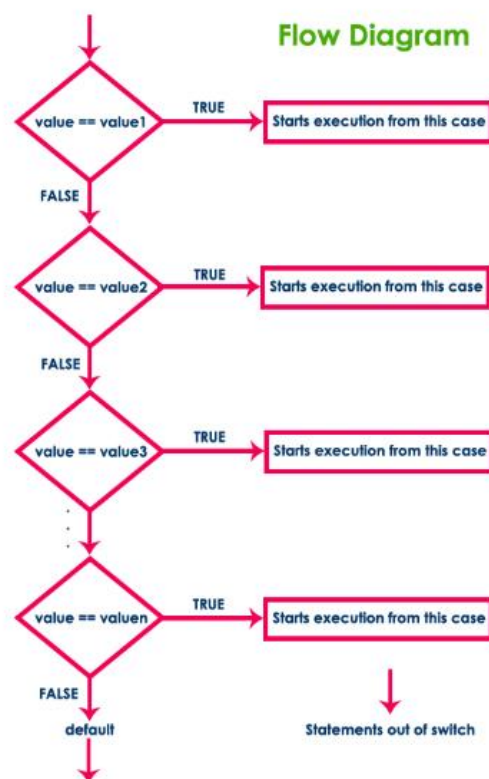
Using the switch statement, one can select only one option from more number of options very easily. In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option. In the switch statement, every option is defined as a **case**.

The switch statement has the following syntax and execution flow diagram.

Syntax

```
switch ( expression or value )  
{  
    case value1: set of statements;  
        ....  
    case value2: set of statements;  
        ....  
    case value3: set of statements;  
        ....  
    case value4: set of statements;  
        ....  
    case value5: set of statements;  
        ....  
    .  
    .  
    default: set of statements;  
}
```

Flow Diagram



Let's look at the following example java code.

```
import java.util.Scanner;

public class SwitchStatementTest {

    public static void main(String[] args) {

        Scanner read = new Scanner(System.in);
        System.out.print("Press any digit: ");

        int value = read.nextInt();

        switch( value )
        {
            case 0: System.out.println("ZERO") ; break ;
            case 1: System.out.println("ONE") ; break ;
            case 2: System.out.println("TWO") ; break ;
            case 3: System.out.println("THREE") ; break ;
            case 4: System.out.println("FOUR") ; break ;
            case 5: System.out.println("FIVE") ; break ;
            case 6: System.out.println("SIX") ; break ;
            case 7: System.out.println("SEVEN") ; break ;
            case 8: System.out.println("EIGHT") ; break ;
            case 9: System.out.println("NINE") ; break ;
            default: System.out.println("Not a Digit") ;
        }

    }

}
```

When we run this code, it produces the following output.



6. Conditional Operators

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 the following example, you can use a *conditional operator* to achieve the same result.

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

The symbols **?** and **:** appearing together is called a conditional operator (also known as a *ternary operator* because it uses three operands. It is the only ternary operator in Java. The conditional operator is in a completely different style, with no explicit **if** in the statement. The syntax to use the operator is as follows:

boolean-expression ? expression1 : expression2

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

Suppose you want to assign the larger number of variable **num1** and **num2** to **max**. You can simply write a statement using the conditional operator:

```
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");
```

As you can see from these examples, the conditional operator enables you to write short and concise code.