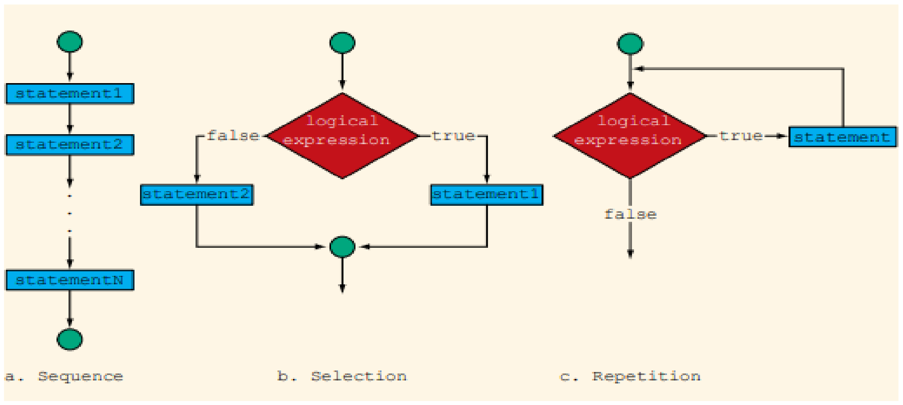
MODULE 5

CONTROL STRUCTURES

Control Structures

A computer can process a program in one of three ways:

- In sequence
- By making a selection or a choice, which is also called a branch
- By repetition, executing a statement over and over using a structure called a loop



Control structures provide alternatives to sequential program execution and are used to alter the flow of execution. The two most common control structures are selection and repetition. **In selection**, the program executes particular statements depending on one or more conditions. **In repetition,** the program repeats particular statements a certain number of times depending on one or more conditions.

Branch: Altering the flow of program execution by making a selection or choice.

Loop: Altering the flow of program execution by the repetition of statement(s).

Selection: if and if...else

Although there are only two logical values, true and false, they are extremely useful because they permit programs to incorporate decision making that alters the processing flow.

Java has three selection or branch control structures: if and if...else statements, and the switch structure. This section discusses how if and if...else statements can be used to create one-way selection, two-way selection, and multiple selections. The switch structure is discussed later in this module.

One-Way Selection

A bank wants to send a notice to a customer if her or his checking account balance falls below the required minimum balance. That is, if the balance is below the required minimum, the bank should send a notice to the customer; otherwise, it should do nothing. Similarly, if the policyholder of an insurance policy is a non-smoker, the company wants to apply a 10% discount to the policy premium. Both of these examples involve one-way selection.

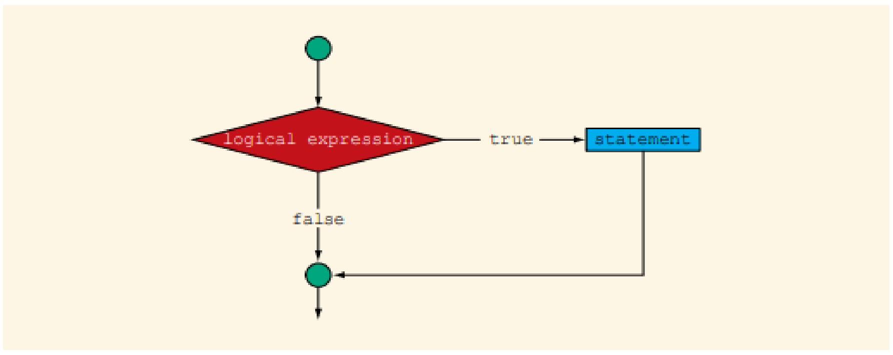
In Java, one-way selections are incorporated using the **if statement**. The **syntax** of one-way selection is:

```
if (expression) {

//statement(s)

}
```

if is a reserved word, followed by the expression enclosed in parentheses. The expression can be a relational or a logical expression that returns either true or false value. If the expression evaluates to true, then the statement between curly braces { and } will execute. If it returns false, the statement will not execute. (Note: The use of curly braces is to enclose the statement(s) to be executed.) For example:

```
int grade = 80;

if (grade >= 70)

{

System.out.println("The student passed.");

}
```

Figure 5-1 shows the flow of execution of the if statement (one-way selection).



Two-Way Selection

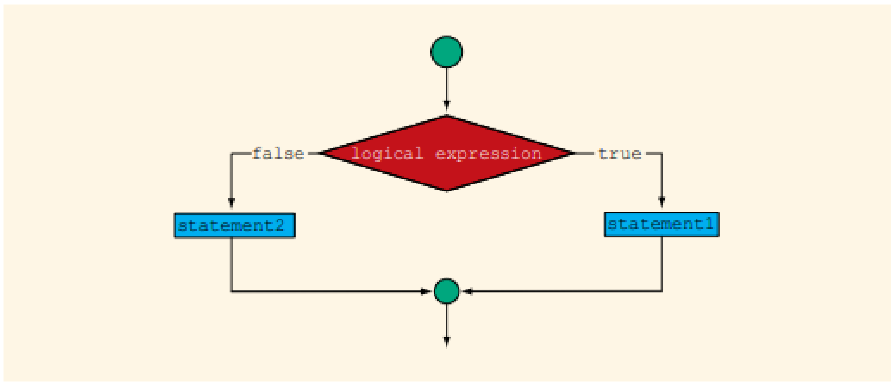
In the previous section, you learned how to implement one-way selections in a program. There are many situations in which you must choose between two alternatives. For example, if a part-time employee works overtime, the paycheck is calculated using the overtime payment formula; otherwise, the paycheck is calculated using the regular formula. This is an example of two-way selection. To choose between two alternatives—that is, to implement two-way selections—Java provides the if...else statement. Two-way selection uses the following syntax:

```
if (expression) {
 //statement 1
 }
else {
//statement 2
}
```

else is a reserved word, which executes when the given expression of if statement evaluates to false. In a two-way selection, if the given expression evaluates to true, then statement 1 executes. If the expression evaluates to false, statement 2 executes. For example:

```
if (grade >= 60) {
 System.out.println("The student passed.");
}
 else {
System.out.println("The student failed.");
 }
```

Figure 5-2 shows the flow of execution ofthe if...else statement (two-way selection).



Compound (Block of) Statements

The if and if...else structures select only one statement at a time. Suppose, however, that you want to execute more than one statement if the logical expression in an if or if...else statement evaluates to true. To permit more complex statements, Java provides a structure called a compound statement or a block of statements. A compound statement takes the following form:

```
{
   statement1
   statement2
       .
       .
       .
   statementn
 }
```

That is, a compound statement or block consists of a sequence of statements enclosed in braces. In an if or if...else structure, a compound statement functions as if it were a single statement. Thus, instead of having a simple two-way selection similar to the following code.

```
if (age > 18)
    System.out.println("Eligible to vote.");
else
    System.out.println("Not eligible to vote.");
```

you could include compound statements, similar to the following code:

```
if (age > 18)
{
    System.out.println("Eligible to vote.");
    System.out.println("No longer a minor.");
}
else
{
    System.out.println("Not eligible to vote.");
    System.out.println("Still a minor.");
}
```

Multiple Selections: Nested if

In the previous sections, you learned how to implement one-way and two-way selections in a program. However, some problems require the implementation of more than two alternatives. For example, suppose that if the checking account balance is greater than or equal to $50000, the interest rate is 5%; if the balance is greater than or equal to $25000 and less than $50000, the interest rate is 4%; if the balance is greater than or equal to $1000 and less than $25000, the interest rate is 3%; otherwise, the interest rate is 0%. This particular problem has four alternatives—that is, multiple selection paths. You can include multiple selection paths in a program by using an if...else structure—if the action statement itself is an if or if...else statement. When one control statement is located within another, it is said to be nested.

Suppose that balance and interestRate are variables of type double. The following statements determine the interestRate depending on the value of balance

```
if (balance >= 50000.00)                         //Line 1
    interestRate = 0.05;                         //Line 2
else                                             //Line 3
    if (balance >= 25000.00)                     //Line 4
        interestRate = 0.04;                     //Line 5
    else                                         //Line 6
        if (balance >= 1000.00)                  //Line 7
            interestRate = 0.03;                 //Line 8
        else                                     //Line 9
            interestRate = 0.00;                 //Line 10
```

Suppose that the value of balance is 60000.00. Then, the expression balance >= 50000.00 in Line 1 evaluates to true and the statement in Line 2 executes. Now suppose the value of balance is 40000.00. Then, the expression balance >= 50000.00 in Line 1 evaluates to false. So the else part at Line 3 executes. The statement part of this else is an if...else statement. Therefore, the expression balance >= 25000.00 is evaluated, which evaluates to true and the statement in Line 5 executes. Note that the expression in Line 4 is evaluated only when the expression in Line 1 evaluates to false. The expression in Line 1 evaluates to false if balance < 50000.00 and then the expression in Line 4 is evaluated. It follows that the expression in Line 4 determines if the value of balance is greater than or equal to 25000 and less than 50000. In other words, the expression in Line 4 is equivalent to the expression (balance >= 25000.00 && balance < 50000.00). The expression in Line 7 works the same way.

The statements illustrate how to incorporate multiple selections using a nested if...else structure.

A nested if...else structure presents an important question: How do you know which else is paired with which if? Recall that in Java there is no stand-alone else statement. Every else must be paired with an if. The rule to pair an else with an if is as follows:

Pairing an else with an if: In a nested if statement, Java associates an else with the most recent incomplete if—that is, the most recent if that has not been paired with an else.

Using this rule, the else in Line 3 is paired with the if in Line 1. The else in Line 6 is paired with the if in Line 4, and the else in Line 9 is paired with the if in Line 7.

To avoid excessive indentation, the code in Example can be rewritten as follows:

```
if (balance >= 50000.00)              //Line 1
    interestRate = 0.05;              //Line 2
else if (balance >= 25000.00)         //Line 3
    interestRate = 0.04;              //Line 4
else if (balance >= 1000.00)          //Line 5
    interestRate = 0.03;              //Line 6
else                                  //Line 7
    interestRate = 0.00;              //Line 8
```

The if statement can be nested inside another if statement or an if…else statement nested within the if statement. A nested if statement allows a program to perform multiple selections. The following are examples of nested if.

Assume that score is a variable of type int. Based on the value of score, the following code determines the grade:

```
if (score >= 90)
    System.out.println("The grade is A");
else if (score >= 80)
    System.out.println("The grade is B");
else if (score >= 70)
    System.out.println("The grade is C");
else if (score >= 60)
    System.out.println("The grade is D");
else
    System.out.println("The grade is F");
```

The following examples will further help you see the various ways in which you can use nested if structures to implement multiple selection.

```
if (temperature >= 50)                              //Line 1
    if (temperature >= 80)                          //Line 2
        System.out.println("Good day for swimming."); //Line 3
    else                                            //Line 4
        System.out.println("Good day for golfing."); //Line 5
else                                                //Line 6
    System.out.println("Good day to play tennis."); //Line 7
```

Short-Circuit Evaluation

Logical expressions in Java are evaluated using an efficient algorithm. This algorithm is illustrated with the help of the following statements:

```
(x > y) || (x == 5)
(a == b) && (x >= 7)
```

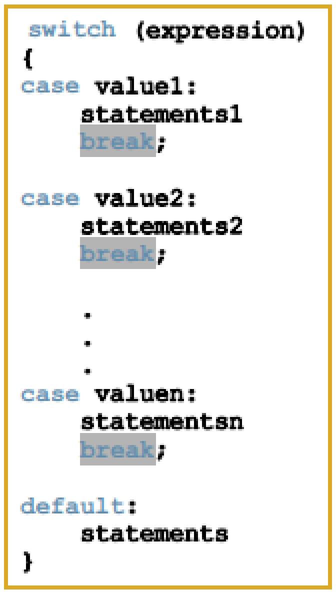
In the first statement, the two operands of the operator || are the expressions (x > y) and (x == 5). This expression evaluates to true if either the operand (x > y) is true or the operand (x == 5) is true. With short-circuit evaluation, the computer evaluates the logical expression from left to right. As soon as the value of the entire logical expression can be determined, the evaluation stops. For example, in the first statement, if the operand (x > y) evaluates to true, then the entire expression evaluates to true because true || true is true and true || false is true. Therefore, the value of the operand (x == 5) has no bearing on the final outcome.

Similarly, in the second statement, the two operands of the operator && are (a == b) and (x >= 7). Now, if the operand (a == b) evaluates to false, then the entire expression evaluates to false because false && true is false and false && false is false.

Short-circuit evaluation (of a logical expression): A process in which the computer evaluates a logical expression from left to right and stops as soon as the value of the expression is determined.

Switch Structures /Statements

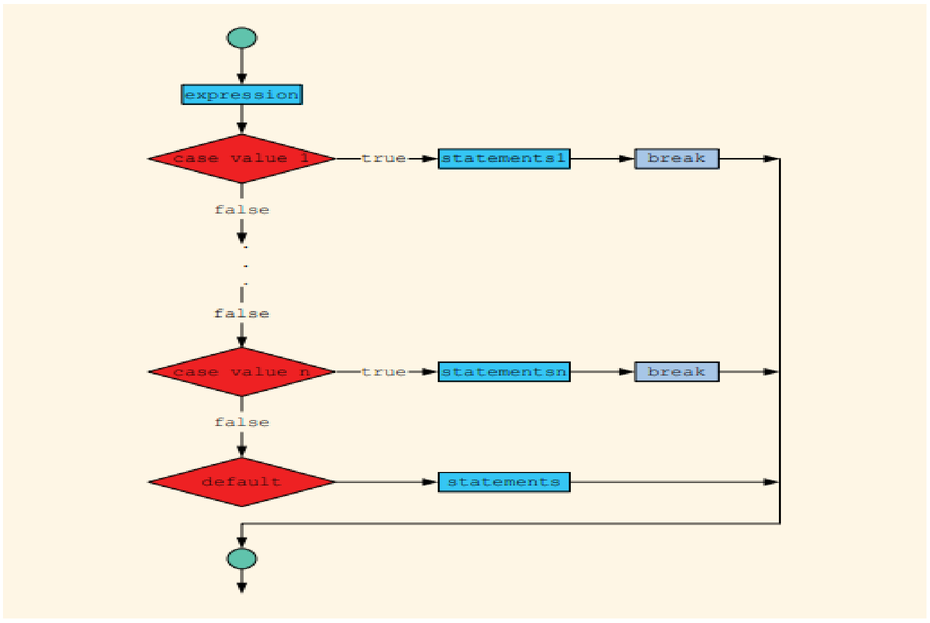
Recall that there are three selection, or branch, structures in Java. The two-selection structure, which is implemented with if and if...else statements, usually requires the evaluation of a (logical) expression. The third selection structure, which does not require the evaluation of a logical expression, is called a switch structure. Java's switch structure gives the computer the power to choose from many alternatives. The general syntax of a switch statement is:



```
switch (expression)
{
case value1:
        statements1
        break;

case value2:
        statements2
        break;

        .
        .
        .
case valuen:
        statementsn
        break;

default:
        statements
}
```

In Java, switch, case, break, and default are reserved words. In a switch structure, the expression is evaluated first. The value of the expression is then used to perform the actions specified in the statements that follow the reserved word case. (Recall that, in a syntax template, the shading indicates an optional part of the definition.)

Although it need not be, the expression is usually an identifier. Whether it is an identifier or an expression, the value of the identifier or the expression can only be of type int, byte, short, or char. The expression is sometimes called the selector. Its value determines which statements are selected for execution. A particular case value must appear only once. One or more statements may follow a case label, so you do not need to use braces to turn multiple statements into a single compound statement. The break statement may or may not appear after each statements1, statements2, ..., statementsn. A switch structure may or may not have the default label. Figure 5.3 shows the flow of execution of a switch statement.



A switch statement executes according to the following rules:

1. When the value of the expression is matched against a case value (also called a label), the statements execute until either a break statement is found or the end of the switch structure is reached.

2. If the value of the expression does not match any of the case values, the statements following the default label execute. If the switch structure has no default label, and if the value of the expression does not match any of the case values, the entire switch statement is skipped.

3. A **break statement** causes an immediate exit from the switch structure.

Consider the following statements (assume that **grade** is a char variable):

```
switch (grade)
{
case 'A':
   System.out.println("The grade is A.");
   break;

case 'B':
   System.out.println("The grade is B.");
   break;

case 'C':
   System.out.println("The grade is C.");
   break;

case 'D':
   System.out.println("The grade is D.");
   break;

case 'F':
   System.out.println("The grade is F.");
   break;

default:
   System.out.println("The grade is invalid.");
}
```

In this example, the expression in the switch statement is a variable identifier. The variable **grade** is of type char, which is an integral type. The valid values of **grade** are **'A'**, **'B'**, **'C'**, **'D'**, and **'F'**. Each case label specifies a different action to take, depending on the value of **grade**. If the value of **grade** is **'A'**, the output is:

```
The grade is A.
```