5.6 `switch` Multiple-Selection Statement

Chapter 4 discussed the `if` single-selection statement and the `if`…`else` double-selection statement. The `switch` **multiple-selection statement** performs different actions based on the possible values of a **constant integral expression** of type `byte`, `short`, `int` or `char` (but not `long`). The expression may also be a `String`, which we demonstrate in Section 5.7.

Using a `switch` Statement to Count A, B, C, D and F Grades

Figure 5.9 calculates the class average of a set of user-entered numeric grades. The program's `switch` statement determines whether each grade is the equivalent of an A, B, C, D or F and increments the appropriate grade counter. The program also displays a summary of the number of students who received each grade.

```
1   // Fig. 5.9: LetterGrades.java
2   // LetterGrades class uses the switch statement
```

```
3   import java.util.Scanner;
4
5   public class LetterGrades {
6     public static void main(String[] args) {
7        int total = 0; // sum of grades
8        int gradeCounter = 0; // number of grades e
9        int aCount = 0; // count of A grades
10       int bCount = 0; // count of B grades
11       int cCount = 0; // count of C grades
12       int dCount = 0; // count of D grades
13       int fCount = 0; // count of F grades
14
15       Scanner input = new Scanner(System.in);
16
17       System.out.printf("%s%n%s%n %s%n %s%n",
18          "Enter the integer grades in the range 0
19          "Type the end-of-file indicator to termi
20          "On UNIX/Linux/macOS type <Ctrl> d then
21          "On Windows type <Ctrl> z then press Ent
22
23       // loop until user enters the end-of-file i
24       while (input.hasNext()) {
25          int grade = input.nextInt(); // read gra
26          total += grade; // add grade to total
27          ++gradeCounter; // increment number of g
28
29          // increment appropriate letter-grade co
30          switch (grade / 10) {
31             case 9: // grade was between
32             case 10: // and 100, inclusive
33                ++aCount;
34                break; // exits switch
35             case 8: // grade was between 80 and 8
36                ++bCount;
37                break; // exits switch
38             case 7: // grade was between 70 and 7
39                ++cCount;
40                break; // exits switch
41             case 6: // grade was between 60 and 6
42                ++dCount;
```

```
                    break; // exits switch
                default: // grade was less than 60
                    ++fCount;
                    break; // optional; exits switch a
                }
            }

        // display grade report
        System.out.printf("%nGrade Report:%n");

        // if user entered at least one grade…
        if (gradeCounter != 0) {
            // calculate average of all grades enter
            double average = (double) total / gradeC

            // output summary of results
            System.out.printf("Total of the %d grades
                gradeCounter, total);
            System.out.printf("Class average is %.2f%
            System.out.printf("%n%s%n%s%d%n%s%d%n%s%d
            "Number of students who received each gra
            "A: ", aCount, // display number of A gra
            "B: ", bCount, // display number of B gra
            "C: ", cCount, // display number of C gra
            "D: ", dCount, // display number of D gra
            "F: ", fCount); // display number of F gr
        }
        else { // no grades were entered, so output
            System.out.println("No grades were entere
        }
    }
}
```

```
Enter the integer grades in the range 0-100.
Type the end-of-file indicator to terminate input:
   On UNIX/Linux/macOS type <Ctrl> d then press Enter
   On Windows type <Ctrl> z then press Enter
99
```

```
92
45
57
63
71
76
85
90
100
^Z

Grade Report:
Total of the 10 grades entered is 778
Class average is 77.80

Number of students who received each grade: A: 4
B: 1
C: 2
D: 1
F: 2
```

Fig. 5.9

LetterGrades class uses the switch statement to count letter grades.

Like earlier versions of the class-average program, the `main` method of class `Letter-Grades` (Fig. 5.9) declares local variables `total` (line 7) and `gradeCounter` (line 8) to keep track of the sum of the grades entered by the user and the number of grades entered, respectively. Lines 9–13 declare counter variables for each grade category. Note that the variables in lines 7–13 are explicitly initialized to `0`.

Method `main` has two key parts. Lines 24–48 read an arbitrary number of integer grades from the user using sentinel-controlled iteration, update instance variables `total` and `gradeCounter`, and increment an appropriate letter-grade counter for each grade entered. Lines 51–72 output a report containing the total of all grades entered, the average of the grades and the number of students who received each letter grade. Let's examine these parts in more detail.

Reading Grades from the User

Lines 17–21 prompt the user to enter integer grades and to type the end-of-file indicator to terminate the input. The **end-of-file indicator** is a system-dependent keystroke combination which the user enters to indicate that there's *no more data to input*. In Chapter 15, Files, Input/Output Streams, NIO and XML Serialization, you'll see how the end-of-file indicator is used when a program reads its input from a file.

On UNIX/Linux/macOS systems, end-of-file is entered by typing the sequence

```
<Ctrl> d
```

on a line by itself. This notation means to simultaneously press both the *Ctrl* key and the *d* key. On Windows systems, end-of-file can be entered by typing

```
<Ctrl> z
```

[*Note:* On some systems, you must press *Enter* after typing the end-of-file key sequence. Also, Windows typically displays the characters `^Z` on the screen when the end-of-file indicator is typed, as shown in the output of Fig. 5.9.]

Portability Tip 5.1

The keystroke combinations for entering end-of-file are system dependent.

The `while` statement (lines 24–48) obtains the user input. The condition at line 24 calls `Scanner` method `hasNext` to determine whether there's more data to input. This method returns the `boolean` value `true` if there's more data; otherwise, it returns `false`. The returned value is then used as the value of the condition in the `while` statement. Method `hasNext` returns `false` once the user types the end-of-file indicator.

Line 25 inputs a grade value from the user. Line 26 adds `grade` to `total`. Line 27 increments `gradeCounter`. These variables are used to compute the average of the grades. Lines 30–47 use a `switch` statement to increment the appropriate letter-grade counter based on the numeric grade entered.

Processing the Grades

The `switch` statement (lines 30–47) determines which counter to increment. We assume that the user enters a valid grade in the range 0–100. A grade in the range 90–100 represents A, 80–89 represents B, 70–79 represents C, 60–69 represents D and 0–59 represents F. The `switch` statement consists of a block that contains a sequence of `case` **labels** and an optional `default` **case**. These are used in this example to determine which counter to increment based on the grade.

When the flow of control reaches the `switch`, the program evaluates the expression in the parentheses (`grade / 10`) following keyword `switch`. This is the `switch`'s **controlling expression**. The program compares this expression's value (which must evaluate to an integral value of type `byte`, `char`, `short` or `int`, or to a `String`) with each `case` label. The controlling expression in line 30 performs integer division, which *truncates the fractional part* of the result. Thus, when we divide a value from 0 to 100 by 10, the result is always a value from 0 to 10. We use several of these values in our `case` labels. For example, if the user enters the integer `85`, the controlling expression evaluates to 8. The `switch` compares `8` with each `case` label. If a match occurs (`case 8:` at line 35), the program executes that `case`'s statements. For the integer `8`, line 36 increments `bCount`, because a grade in the 80s is a B. The `break` **statement** (line 37) causes program control to proceed with the first statement after the `switch`—in this program, we reach the end of the

`while` loop, so control returns to the loop-continuation condition in line 24 to determine whether the loop should continue executing.

The `case`s explicitly test for the values `10`, `9`, `8`, `7` and `6`. Note the cases at lines 31–32 that test for the values `9` and `10` (both of which represent the grade A). Listing cases consecutively in this manner with no statements between them enables the cases to perform the same set of statements—when the controlling expression evaluates to `9` or `10`, the statements in lines 33–34 will execute. The `switch` statement does *not* provide a mechanism for testing *ranges* of values, so *every* value you need to test must be listed in a separate `case` label. Each `case` can have multiple statements. The `switch` statement differs from other control statements in that it does *not* require braces around multiple statements in a `case`.

`case` without a `break` Statement

Without `break` statements, each time a match occurs in the `switch`, the statements for that case and subsequent cases execute until a `break` statement or the end of the `switch` is encountered. This is often referred to as “falling through” to the statements in subsequent `case`s. (This feature is perfect for writing a concise program that displays the iterative song “The Twelve Days of Christmas” in Exercise 5.29.)

Common Programming Error 5.7

Forgetting a `break` *statement when one is needed in a* `switch` *is a logic error.*

The `default` Case

If no match occurs between the controlling expression's value and a `case` label, the `default` case (lines 44–46) executes. We use the `default` case in this example to process all controlling-expression values that are less than `6`—that is, all failing grades. If no match occurs and the `switch` does not contain a `default` case, program control simply continues with the first statement after the `switch`.

Error-Prevention Tip 5.9

In a `switch` *statement, ensure that you test all possible values of the controlling expression.*

Displaying the Grade Report

Lines 51–72 output a report based on the grades entered (as

shown in the input/output window in Fig. 5.9). Line 54 determines whether the user entered at least one grade—this helps us avoid dividing by zero. If so, line 56 calculates the average of the grades. Lines 59–68 then output the total of all the grades, the class average and the number of students who received each letter grade. If no grades were entered, line 71 outputs an appropriate message. The output in Fig. 5.9 shows a sample grade report based on 10 grades.

`switch` Statement UML Activity Diagram

Figure 5.10 shows the UML activity diagram for the general `switch` statement. Most `switch` statements use a `break` in each `case` to terminate the `switch` statement after processing the `case`. Figure 5.10 emphasizes this by including `break` statements in the activity diagram. The diagram makes it clear that the `break` statement at the end of a `case` causes control to exit the `switch` statement immediately.

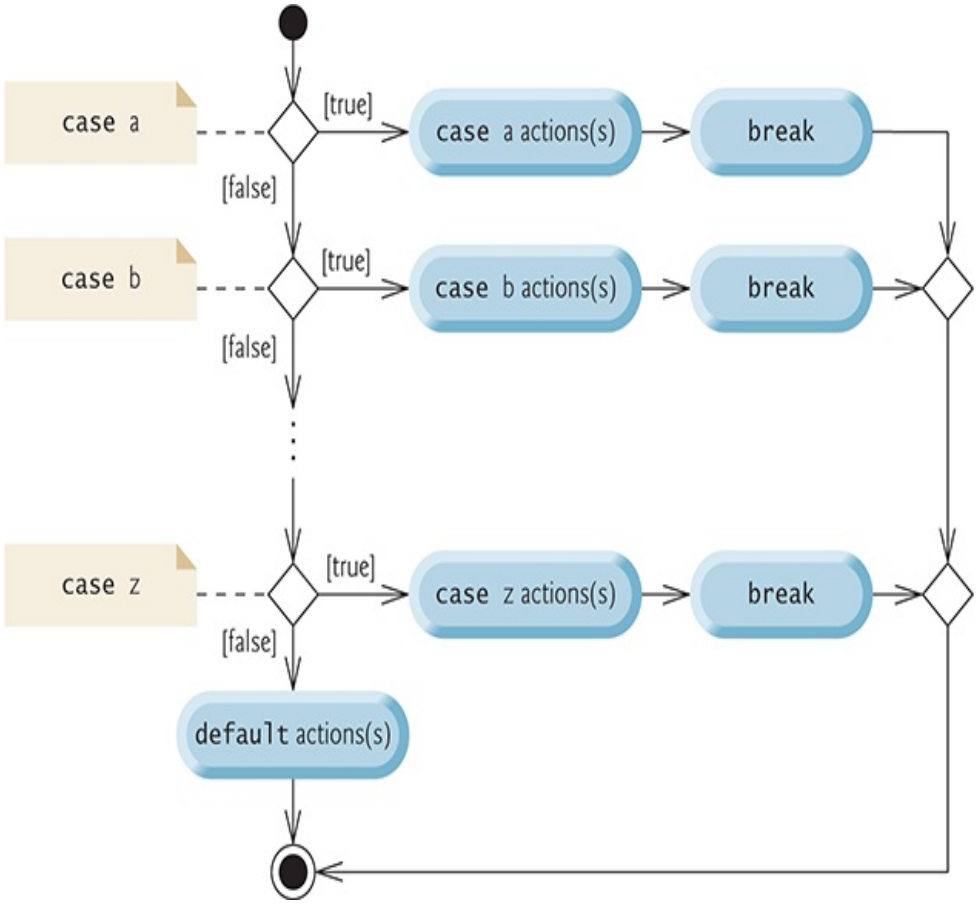

Fig. 5.10

`switch` multiple-selection statement UML activity diagram with `break` statements.

Description

The `break` statement is *not* required for the `switch`'s last `case` (or the optional `default` case, when it appears last), because execution continues with the next statement after the `switch`.

Error-Prevention Tip 5.10

Provide a `default` *case in* `switch` *statements. This focuses you on the need to process exceptional conditions.*

Good Programming Practice 5.2

Although each `case` *and the* `default` *case in a* `switch` *can occur in any order, place the* `default` *case last. When the* `default` *case is last, the* `break` *for that case is not required.*

Notes on the Expression in Each `case` of a `switch`

When using a `switch`, remember that each `case` must contain a `String` or a constant integral expression—that is, any combination of integer constants that evaluates to a constant integer value (e.g., –7, 0 or 221). An integer constant is simply an integer value. In addition, you can use **character constants**—specific characters in single quotes, such as `'A'`, `'7'` or `'$'`—which represent the integer values of characters.

(Appendix B shows the integer values of the characters in the ASCII character set, which is a subset of the Uni-code® character set used by Java.)

The expression in each `case` can also be a **constant variable**—a variable containing a value which does not change for the entire program. Such a variable is declared with keyword `final` (discussed in Chapter 6). Java has a feature called `enum` types, which we also present in Chapter 6—`enum` type constants can also be used in `case` labels.

In Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces, we present a more elegant way to implement `switch` logic—we use a technique called *polymorphism* to create programs that are often clearer, easier to maintain and easier to extend than programs using `switch` logic.