In this book, the Java programming concepts and techniques are presented in a logical order. When these concepts and techniques are learned one at a time in a logical order, they are simple enough to be understood completely. Understanding a concept or technique completely before using it will save you an enormous amount of debugging time.

Program Style and Form (Revisited): Indentation

In the section "Program Style and Form" of Chapter 2, we specified some guidelines to write programs. Now that we have started discussing control structures, in this section, we give some general guidelines to properly indent your program.

As you write programs, typos and errors are unavoidable. If your program is properly indented, you can spot and fix errors quickly as shown by several examples in this chapter. Typically, the IDE that you use will automatically indent your program. If for some reason your IDE does not indent your program, you can indent your program yourself.

Proper indentation can show the natural grouping of statements. You should insert a blank line between statements that are naturally separate. In this book, the statements inside braces, the statements of selection structures, an `if` statement within an `if` statement are all indented four spaces to the right. Throughout the book, we use four spaces of indentation for statements; we especially use indentation to show the level of a control structure within another control structure. You can also use four spaces for indentation.

There are two commonly used styles for placing braces. In this book, we place braces on a line by themselves. Also, matching left and right braces are in the same column, that is, they are the same number of spaces away from the left side of the program. This style of placing braces easily shows the grouping of the statements as well as matching left and right braces. You can also follow this style to place and indent braces.

In the second style of placing braces, the left brace need not be on a line by itself. Typically, for control structures, the left brace is placed after the last right parenthesis of the (logical) expression and the right brace is on a line by itself. This style might save some space. However, sometimes this style might not immediately show the grouping or the block of the statements.

No matter what style of indentation you use, you should be consistent within your programs and the indentation should show the structure of the program.

`switch` Structures

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 4-5 shows the flow of execution of a **switch** statement.

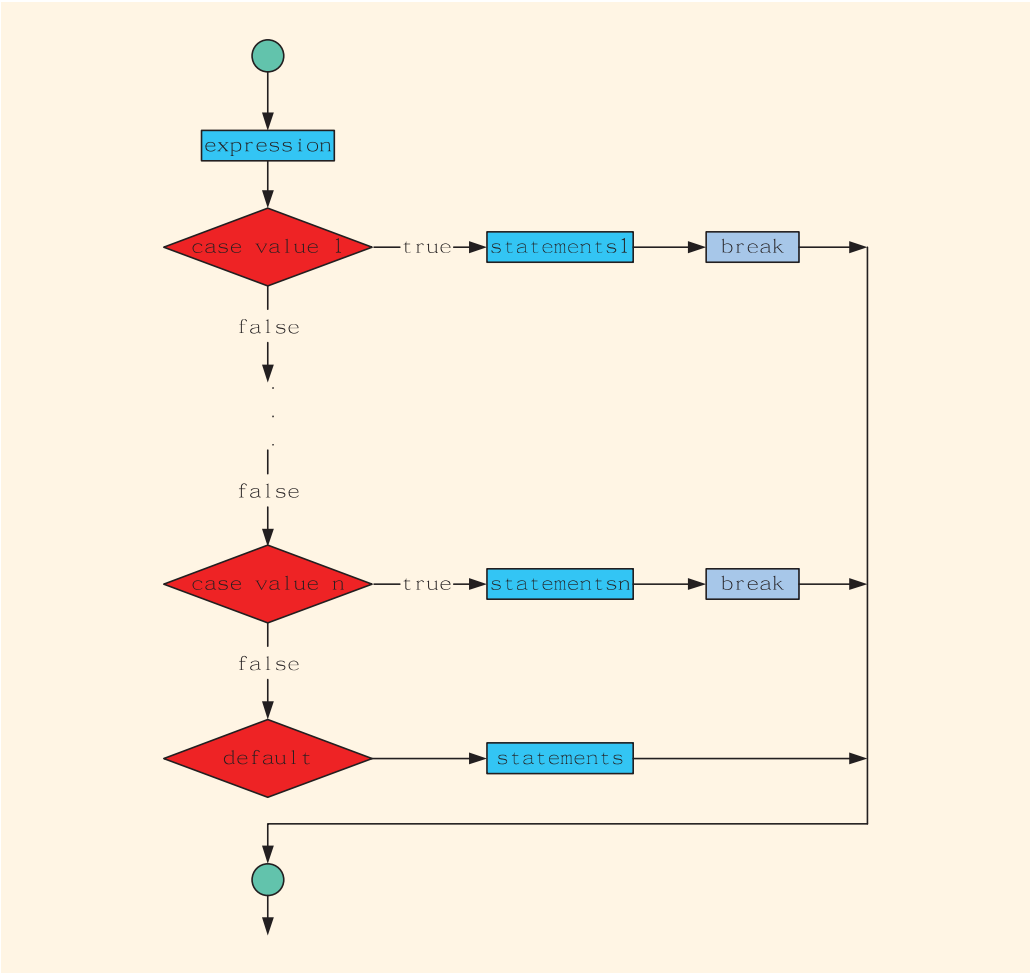

FIGURE 4-5 `switch` statement

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.

EXAMPLE 4-20

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.
```

EXAMPLE 4-21

The following program illustrates the effect of the break statement. It asks the user to input a number between 0 and 10.

```
//Effect of break statements in a switch structure

import java.util.*;

public class BreakStatementsInSwitch
{
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        int num;

        System.out.print("Enter an integer between "
                       + "0 and 10: ");              //Line 1
        num = console.nextInt();                     //Line 2
        System.out.println();                        //Line 3

        System.out.println("The number you entered "
                        + "is " + num);              //Line 4

        switch (num)                                 //Line 5
        {
        case 0:                                      //Line 6
        case 1:                                      //Line 7
            System.out.print("Hello ");               //Line 8
        case 2:                                      //Line 9
            System.out.print("there. ");              //Line 10
        case 3:                                      //Line 11
            System.out.print("I am ");                //Line 12
        case 4:                                      //Line 13
            System.out.println("Mickey.");             //Line 14
            break;                                   //Line 15

        case 5:                                      //Line 16
            System.out.print("How ");                 //Line 17
        case 6:                                      //Line 18
        case 7:                                      //Line 19
        case 8:                                      //Line 20
             System.out.println("are you?");           //Line 21
             break;                                  //Line 22

        case 9:                                      //Line 23
            break;                                   //Line 24

        case 10:                                     //Line 25
            System.out.println("Have a nice day.");    //Line 26
            break;                                   //Line 27
        default:                                     //Line 28
            System.out.println("Sorry the number is "
                             + "out of range.");     //Line 29
        }
```

```
        System.out.println("Out of switch "
                           + "structure.");            //Line 30
    }
}
```

Sample Runs

These outputs were obtained by executing the preceding program several times. In each of these outputs, the user input is shaded.

Sample Run 1:

```
Enter an integer between 0 and 10: 0

The number you entered is 0
Hello there. I am Mickey.
Out of switch structure.
```

Sample Run 2:

```
Enter an integer between 0 and 10: 3

The number you entered is 3
I am Mickey.
Out of switch structure.
```

Sample Run 3:

```
Enter an integer between 0 and 10: 4

The number you entered is 4
Mickey.
Out of switch structure.
```

Sample Run 4:

```
Enter an integer between 0 and 10: 7

The number you entered is 7
are you?
Out of switch structure.
```

Sample Run 5:

```
Enter an integer between 0 and 10: 9

The number you entered is 9
Out of switch structure.
```

A walk-through of this program, using certain values of the `switch` expression `num`, can help you understand how the `break` statement functions. If the value of `num` is `0`, the value of the `switch` expression matches the `case` value `0`. All statements following `case 0:` execute until a `break` statement appears.

The first `break` statement appears at Line 15, just before the `case` value of `5`. Even though the value of the `switch` expression does not match any of the `case` values (`1`, `2`, `3`, or `4`), the statements following these values execute.

When the value of the `switch` expression matches a `case` value, *all* statements execute until a `break` is encountered, and the program skips all `case` labels in between. Similarly, if the value of `num` is `3`, it matches the `case` value of `3` and the statements following this label execute until the `break` statement is encountered at Line 15. If the value of `num` is `9`, it matches the `case` value of `9`. In this situation, the action is empty, because only the `break` statement, at Line 24, follows the `case` value of `9`.

EXAMPLE 4-22

Although a `switch` structure's `case` values (labels) are limited, the `switch` statement `expression` can be as complex as necessary. Consider the following `switch` statement:

```
switch (score / 10)
{
case 0:
case 1:
case 2:
case 3:
case 4:
case 5:
    grade = 'F';
    break;

case 6:
    grade = 'D';
    break;

case 7:
    grade = 'C';
    break;

case 8:
    grade = 'B';
    break;

case 9:
case 10:
    grade = 'A';
    break;

default:
    System.out.println("Invalid test score.");
}
```

Assume that `score` is an `int` variable with values between `0` and `100`. If `score` is `75`, then `score / 10 = 75 / 10 = 7` and the grade assigned is `'C'`. If the value of `score` is between `0` and `59`, then the grade is `'F'`. If `score` is between `0` and `59`, `score / 10` is `0`, `1`, `2`, `3`, `4`, or `5`; each of these values corresponds to the grade `'F'`.

Therefore, in this `switch` structure, the action statements of `case 0`, `case 1`, `case 2`, `case 3`, `case 4`, and `case 5` are all the same. Rather than write the statement `grade = 'F';` followed by the `break` statement for each of the `case` values of `0`, `1`, `2`, `3`, `4`, and `5`, you can simplify the programming code by first specifying all of the case values (as shown in the preceding code) and then specifying the desired action statement. The `case` values of `9` and `10` follow similar conventions.

CHOOSING BETWEEN AN `if...else` AND A `switch` STRUCTURE

As you can see from the preceding examples, the `switch` statement is an elegant way to implement multiple selections. You will see a `switch` statement used in the programming examples in this chapter. There are no fixed rules that can be applied to decide whether to use an `if...else` structure or a `switch` structure to implement multiple selections, but you should remember the following consideration: If multiple selections involve a range of values, you should use either an `if...else` structure or a `switch` structure wherein you convert each range to a finite set of values.

For instance, in Example 4-22, the value of `grade` depends on the value of `score`. If `score` is between `0` and `59`, `grade` is `'F'`. Because `score` is an `int` variable, 60 values correspond to the grade of `'F'`. If you list all 60 values as `case` values, the `switch` statement could be very long. However, dividing by 10 reduces these 60 values to only 6 values: `0`, `1`, `2`, `3`, `4`, and `5`.

If the range of values is infinite and you cannot reduce them to a set containing a finite number of values, you must use the `if...else` structure. For example, suppose that `score` is a `double` variable. The number of `double` values between `0` and `60` is (practically) infinite. However, you can use the expression `(int)(score) / 10` and reduce the infinite number of values to just six values.

DEBUGGING

Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques (Revisited)

Earlier in this chapter, we discussed how a partial understanding of a concept or technique can lead to errors in a program. In this section, we give another example to illustrate the problem of using partially understood concepts and techniques. In Example 4-22, we illustrated how to assign a grade based on a test score between 0 and 100. Next consider the following program that assigns a grade based on a test score.